



Open Source Development, Communities and Quality

WCC 2008 Milano, Italy

Edited by
Barbara Russo
Ernesto Damiani
Scott Hissam
Björn Lundell
Giancarlo Succi

 Springer




IFIP
WCC 2008
WORLD COMPUTER CONGRESS
Milano

**OPEN SOURCE DEVELOPMENT,
COMMUNITIES AND QUALITY**

IFIP – The International Federation for Information Processing

IFIP was founded in 1960 under the auspices of UNESCO, following the First World Computer Congress held in Paris the previous year. An umbrella organization for societies working in information processing, IFIP's aim is two-fold: to support information processing within its member countries and to encourage technology transfer to developing nations. As its mission statement clearly states,

IFIP's mission is to be the leading, truly international, apolitical organization which encourages and assists in the development, exploitation and application of information technology for the benefit of all people.

IFIP is a non-profitmaking organization, run almost solely by 2500 volunteers. It operates through a number of technical committees, which organize events and publications. IFIP's events range from an international congress to local seminars, but the most important are:

- The IFIP World Computer Congress, held every second year;
- Open conferences;
- Working conferences.

The flagship event is the IFIP World Computer Congress, at which both invited and contributed papers are presented. Contributed papers are rigorously refereed and the rejection rate is high.

As with the Congress, participation in the open conferences is open to all and papers may be invited or submitted. Again, submitted papers are stringently refereed.

The working conferences are structured differently. They are usually run by a working group and attendance is small and by invitation only. Their purpose is to create an atmosphere conducive to innovation and development. Refereeing is less rigorous and papers are subjected to extensive group discussion.

Publications arising from IFIP events vary. The papers presented at the IFIP World Computer Congress and at open conferences are published as conference proceedings, while the results of the working conferences are often published as collections of selected and edited papers.

Any national society whose primary activity is in information may apply to become a full member of IFIP, although full membership is restricted to one society per country. Full members are entitled to vote at the annual General Assembly, National societies preferring a less committed involvement may apply for associate or corresponding membership. Associate members enjoy the same benefits as full members, but without voting rights. Corresponding members are not represented in IFIP bodies. Affiliated membership is open to non-national societies, and individual and honorary membership schemes are also offered.

OPEN SOURCE DEVELOPMENT, COMMUNITIES AND QUALITY

*IFIP 20th World Computer Congress,
Working Group 2.3 on Open Source Software,
September 7-10, 2008, Milano, Italy*

Edited by

Barbara Russo

*Free University of Bolzano-Bozen
Italy*

Ernesto Damiani

*University of Milano-Bicocca
Italy*

Scott Hissam

*Carnegie Mellon Software
Engineering Institute
USA*

Björn Lundell

*University of Skövde
Sweden*

Giancarlo Succi

*Free University of Bolzano-Bozen
Italy*

Library of Congress Control Number: 2008929506

Open Source Development, Communities and Quality

Edited by Barbara Russo, Ernesto Damiani, Scott Hissam,
Björn Lundell and Giancarlo Succi

p. cm. (IFIP International Federation for Information Processing, a Springer Series
in Computer Science)

ISSN: 1571-5736 / 1861-2288 (Internet)

ISBN: 978-0-387-09683-4

eISBN: 978-0-387-09684-1

Printed on acid-free paper

Copyright © 2008 by International Federation for Information Processing.

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

9 8 7 6 5 4 3 2 1

springer.com

IFIP 2008 World Computer Congress (WCC'08)

Message from the Chairs

Every two years, the International Federation for Information Processing hosts a major event which showcases the scientific endeavours of its over one hundred Technical Committees and Working Groups. 2008 sees the 20th World Computer Congress (WCC 2008) take place for the first time in Italy, in Milan from 7-10 September 2008, at the MIC - Milano Convention Centre. The Congress is hosted by the Italian Computer Society, AICA, under the chairmanship of Giulio Occhini.

The Congress runs as a federation of co-located conferences offered by the different IFIP bodies, under the chairmanship of the scientific chair, Judith Bishop. For this Congress, we have a larger than usual number of thirteen conferences, ranging from Theoretical Computer Science, to Open Source Systems, to Entertainment Computing. Some of these are established conferences that run each year and some represent new, breaking areas of computing. Each conference had a call for papers, an International Programme Committee of experts and a thorough peer reviewed process. The Congress received 661 papers for the thirteen conferences, and selected 375 from those representing an acceptance rate of 56% (averaged over all conferences).

An innovative feature of WCC 2008 is the setting aside of two hours each day for cross-sessions relating to the integration of business and research, featuring the use of IT in Italian industry, sport, fashion and so on. This part is organized by Ivo De Lotto. The Congress will be opened by representatives from government bodies and Societies associated with IT in Italy.

This volume is one of fourteen volumes associated with the scientific conferences and the industry sessions. Each covers a specific topic and separately or together they form a valuable record of the state of computing research in the world in 2008. Each volume was prepared for publication in the Springer IFIP Series by the conference's volume editors. The overall Chair for all the volumes published for the Congress is John Impagliazzo.

For full details on the Congress, refer to the webpage <http://www.wcc2008.org>.

Judith Bishop, South Africa, Co-Chair, International Program Committee
Ivo De Lotto, Italy, Co-Chair, International Program Committee
Giulio Occhini, Italy, Chair, Organizing Committee
John Impagliazzo, United States, Publications Chair

WCC 2008 Scientific Conferences

TC12	AI	Artificial Intelligence 2008
TC10	BICC	Biologically Inspired Cooperative Computing
WG 5.4	CAI	Computer-Aided Innovation (Topical Session)
WG 10.2	DIPES	Distributed and Parallel Embedded Systems
TC14	ECS	Entertainment Computing Symposium
TC3	ED_L2L	Learning to Live in the Knowledge Society
WG 9.7	HCE3	History of Computing and Education 3
TC3		
TC13	HCI	Human Computer Interaction
TC8	ISREP	Information Systems Research, Education and Practice
WG 12.6	KMIA	Knowledge Management in Action
TC2	OSS	Open Source Systems
WG 2.13		
TC11	IFIP SEC	Information Security Conference
TC1	TCS	Theoretical Computer Science

IFIP

- is the leading multinational, apolitical organization in Information and Communications Technologies and Sciences
- is recognized by United Nations and other world bodies
- represents IT Societies from 56 countries or regions, covering all 5 continents with a total membership of over half a million
- links more than 3500 scientists from Academia and Industry, organized in more than 101 Working Groups reporting to 13 Technical Committees
- sponsors 100 conferences yearly providing unparalleled coverage from theoretical informatics to the relationship between informatics and society including hardware and software technologies, and networked information systems

Details of the IFIP Technical Committees and Working Groups can be found on the website at <http://www.ifip.org>.

Contents

WCC 2008 Congress..... v

Contents vii

Preface..... xiii

About IFIP WG 2.13 xiv

Acknowledgements xv

OSS 2008 Conference Organization xvii

List of Contributors xix

Part I Full Papers

A Framework for Evaluating Managerial Styles in Open Source Projects 1
Eugenio Capra and Anthony I. Wasserman

Forging a Community – Not: Experiences on Establishing an Open Source Project 15
Juha Järvensivu and Tommi Mikkonen

Mapping Linux Security Targets to Existing Test Suites 29
C.A. Ardagna, E. Damiani, N. El Ioini, F. Frati, P. Giovannini and R. Tchokpon

Overview on Trust in Large FLOSS Communities 47
Etiel Petrinja, Alberto Sillitti and Giancarlo Succi

PMLite: An Open Source Solution for Process Monitoring..... 57
Alberto Colombo, Ernesto Damiani and Fulvio Frati

Requirements Acquisition in Open Source Development: Firefox 2.0 69
John Noll

Analysis of Coordination between Developers and Users in the Apache Community.....	81
<i>Yasutaka Kamei, Shinsuke Matsumoto, Hirotaka Maeshima, Yoji Onishi, Masao Ohira and Ken-ichi Matsumoto</i>	
Lost and Gained in Translation: Adoption of Open Source Software Development at Hewlett-Packard.....	93
<i>Catharina Melian and Magnus Mähring</i>	
Mining for Practices in Community Collections: Finds From Simple Wikipedia.....	105
<i>Matthijs den Besten, Alessandro Rossi, Loris Gaio, Max Loubser and Jean-Michel Dalle</i>	
Open To Grok. How Do Hackers' Practices Produce Hackers?	121
<i>Vincenzo D'Andrea, Stefano De Paoli and Maurizio Teli</i>	
Social Dynamics of FLOSS Team Communication across Channels.....	131
<i>Andrea Wiggins, James Howison and Kevin Crowston</i>	
Towards a Global Research Infrastructure for Multidisciplinary Study of Free/Open Source Software Development.....	143
<i>Les Gasser and Walt Scacchi</i>	
Update Propagation Practices in Highly Reusable Open Source Components	159
<i>Heikki Orsila, Jaco Geldenhuys, Anna Ruokonen and Imed Hammouda</i>	
Using Social Network Analysis Techniques to Study Collaboration between a FLOSS Community and a Company	171
<i>Juan Martinez-Romo, Gregorio Robles, Jesus M. Gonzalez-Barahona and Miguel Ortuño-Perez</i>	
Empirical Analysis of the Bug Fixing Process in Open Source Projects	187
<i>Chiara Francalanci and Francesco Merlo</i>	
The Total Growth of Open Source.....	197
<i>Amit Deshpande and Dirk Riehle</i>	
Adoption of Open Source in the Software Industry	211
<i>Øyvind Hauge, Carl-Fredrik Sørensen and Reidar Conradi</i>	
Migration Discourse Structures: Escaping Microsoft's Desktop Path	223
<i>Leonhard Dobusch</i>	

The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation	237
<i>Ioannis Samoladas, Georgios Gousios, Diomidis Spinellis and Ioannis Stamelos</i>	

Part II Short Papers

An Open Integrated Environment for Transparent Fuzzy Agents Design.....	249
<i>Giovanni Acampora and Vincenzo Loia</i>	
Archetypal Internet-Scale Source Code Searching	257
<i>Medha Umarji, Susan Elliott Sim and Crista Lopes</i>	
Channeling Firefox Developers: Mom and Dad Aren't Happy Yet.....	265
<i>Jean-Michel Dalle, Matthijs den Besten and H�la Masmoudi</i>	
Continuous Integration in Open Source Software Development	273
<i>Amit Deshpande and Dirk Riehle</i>	
Extracting Generally Applicable Patterns from Object-Oriented Programs for the Purpose of Test Case Creation	281
<i>Richard Torkar, Robert Feldt and Tony Gorschek</i>	
Social Networking Technologies for Free-Open Source e-Learning Systems	289
<i>Francesco Di Cerbo, Gabriella Dodero and Giancarlo Succi</i>	
The Networked Forge: New Environments for Libre Software Development.....	299
<i>Jesus M. Gonzalez-Barahona, Andr�s Mart�nez, Alvaro Polo, Juan Jos� Hierro, Marcos Reyes, Javier Soriano and Rafael Fern�ndez</i>	
To What Extent Does It Pay To Approach Open Source Software for A Big Telco Player?.....	307
<i>Massimo Banzi, Guido Bruno and Giovanni Caire</i>	
A Framework to Abstract the Design Practices of e-Learning System Projects ..	317
<i>Alain Corbiere</i>	
Assessing Innovation in the Software Sector: Proprietary vs. FOSS Production Mode. Preliminary Evidence from the Italian Case	325
<i>Dario Lorenzi and Cristina Rossi</i>	

Detecting Agility of Open Source Projects Through Developer Engagement	333
<i>Paul J. Adams, Andrea Capiluppi and Adriaan de Groot</i>	
Facilitating Social Network Studies of FLOSS using the OSSNetwork Environment.....	343
<i>Marco A. Balieiro, Samuel F. de Sousa Júnior and Cleidson R. B. de Souza</i>	
Reflection on Knowledge Sharing in F/OSS Projects	351
<i>Sulayman K. Sowe and Ioannis Stamelos</i>	
Usability in Company Open Source Software Context - Initial Findings from an Empirical Case Study	359
<i>Netta Iivari, Henrik Hedberg and Tanja Kirves</i>	
Willingness to Cooperate Within the Open Source Software Domain.....	367
<i>Pascal Ravesteyn and Gilbert Silvius</i>	
Open Source Project Categorization Based on Growth Rate Analysis and Portfolio Planning Methods	375
<i>Stefan Koch and Volker Stix</i>	
Applying Open Source Development Practices Inside a Company	381
<i>Juho Lindman, Matti Rossi and Pentti Marttiin</i>	
Towards The Evaluation of OSS Trustworthiness: Lessons Learned From The Observation of Relevant OSS Projects.....	389
<i>Davide Taibi, Vieri del Bianco, Davide Dalle Carbonare, Luigi Lavazza and Sandro Morasca</i>	
Open Source Reference Systems for Biometric Verification of Identity	397
<i>Aurélien Mayoue and Dijana Petrovska-Delacrétaz</i>	
eResearch Workflows for Studying Free and Open Source Software Development	405
<i>James Howison, Andrea Wiggins and Kevin Crowston</i>	

Part III Panels

Panel: Opportunities and Risks for Open Source Software in Industry	413
<i>Joseph Feller, Björn Lundell, Pentti Marttiin, Walt Scacchi and Nico Schellingerhout</i>	

Part IV Posters and Demonstrations

Open Source Environments for Collaborative Experiments in e-Science 415
*Andrea Bosin, Nicoletta Dessì, Maria Grazia Fugini, Diego Liberati
and Barbara Pes*

eResearch Workflows for Studying Free and Open Source Software
Development 417
James Howison, Andrea Wiggins and Kevin Crowston

Facilitating Social Network Studies of FLOSS using the OSSNetwork
Environment 417
Marco A. Balieiro, Samuel F. de Sousa Júnior and Cleidson R. B. de Souza

Preface

We are very pleased to introduce *Open Source Development, Communities and Quality*. The *International Conference on Open Source Systems* has come to its fourth edition – OSS 2008. Now, Free, Libre, and Open Source software is by all means now one of the most relevant subjects of study in several disciplines, ranging from information technology to social sciences and including also law, business, and political sciences. There are several conference tracks devoted to open source software with several publications appearing in high quality journals and magazines.

OSS 2008 has been organized with the purpose of being **the** reference venue for those working in this area, being the most prominent conference in this area. For this reason OSS 2008 has been located within the frameworks of the 20th World Computer Congress, WCC 2008, in Milan, the largest event of IFIP in 2008.

We believe that this conference series, and the IFIP working group it represents, can play an important role in meeting these challenges, and hope that this book will become a valuable contribution to the open source body of research.

The response of the research community has exceeded our expectations. A significant number of high-quality submission from all parts of the world were carefully peer-reviewed; in the end, 19 papers are published here as full papers (Part I) and 20 published in a condensed form as short papers (Part II). Additionally, 3 submissions presented their work in posters and demonstrations form for discussion at the conference (Part IV). Complementing the paper program, we are pleased to include the description of a panel (Part III) as an engaging and important part of the conference event. In addition to this there are three outstanding keynotes tackling three orthogonal perspectives: the use of Open Source to build a successful business; its adoption in public administration to promote savings, transparency and inclusion; the research done in this area by large public-private consortiums.

Altogether, we believe that this work is an essential milestone in the understanding of OSS and we are confident that the readers of this volume will find it useful and also enjoyable.

About IFIP WG 2.13

The OSS2008 Conference is the fourth conference in this very successful series of conferences on the open source topic. The IFIP Working Group 2.13 on Open Source Software was founded to further consolidate the discipline and to ensure a continuing and professional conference series with an established publisher, and to establish a conference series that would move around the world over time. Proposals for a number of conferences for the next few years have already been received, and anyone who would like to propose a conference theme and venue should come to the next conference and present a formal proposal.

The Working Group has 27 founding members from Europe, North America, Middle East, Asia /Pacific and Africa. The main objective of the Working Group is to enable the diverse community of free, libre and open source software (OSS) researchers and practitioners to rigorously investigate the technology, work practices, development processes, community dynamics within OSS systems, complementing appropriately other IFIP Working Groups where OSS is increasingly relevant. The scope of the Working Group is detailed on the website¹ which is being maintained by Kevin Crowston. Last, becoming a member of the Working Group is open to anyone attending at least two IFIP WG 2.13 meetings. We hope you will join us.

Since the group has been founded, there have been several exciting developments in which group members have been centrally involved. Ongoing projects in which the Working Group's members are central include the following:

- *COSI*² focusing on transferring the lessons of open source and agile methods to traditional in-house development;
- *FOSSRRR*³ designing a global research infrastructure for FLOSS researchers and developers for access to data, artifacts, analyses, and published studies drawn from multi-project FLOSS repositories such as FLOSSmole⁴, the SourceForge Research Data Archive at Notre Dame⁵, and Google Code⁶;
- *OPAALS* (www.opaals.org) seeking to build a sustainable interdisciplinary research community in the area of digital ecosystems;
- *QualiPSo*⁷ studying the quality and trust aspects of Open Source Software and its adoption in the main stream industrial process - in the framework of QualiPSo several competence centers on Open Source are being created around the world;
- *Open Code, Content and Commerce (O3C) Business models*, a three-year investigation of open innovation and value-creation strategies

¹ <http://www.ifipwg213.org>

² www.itea-cosi.org

³ fossrri.rotterdam.ics.uci.edu

⁴ ossmole.sourceforge.net

⁵ zerlot.cse.nd.edu

⁶ google.com/code

⁷ www.qualipso.org

IFIP WG 2.13 OFFICERS

General Chair:	Brian Fitzgerald, Lero, University of Limerick, Ireland
Vice-Chair:	Walt Scacchi, University of California at Irvine, US
Vice-Chair:	Giancarlo Succi, Free University of Bolzano-Bozen, Italy
Secretary:	Ernesto Damiani, University of Milan, Italy

Acknowledgements

We gratefully acknowledge the contributions made by the other OSS 2008 Conference officials: Stefano De Panfilis (Workshop Chair), Alberto Colombo (Workshop Co-Chair and Web Master), and Fulvio Frati (Workshop Co-Chair and Web Master). Special thanks go to Arie Gurfinkel as LaTeX wizard for his aid in preparing this volume. We would like to thank the OSS 2008 international program committee and all the local volunteer organizers and supporters, without whom this conference could not take place. Finally, we thank all the authors, panelists, and workshop organizers who provided this years conference with such an excellent program.

Barbara Russo (Track Chair)⁸
 Ernesto Damiani (General Co-Chair)⁹
 Scott Hissam (Track Chair)¹⁰
 Björn Lundell (Track-Chair)¹¹
 Giancarlo Succi (General Co-Chair)¹²

Milan, Italy
 September 2008

⁸ Free University of Bolzano-Bozen, Italy

⁹ University of Milan, Italy

¹⁰ Carnegie Mellon Software Engineering Institute, Pittsburgh, USA

¹¹ University of Skövde, Sweden

¹² Free University of Bolzano-Bozen, Italy

OSS 2008 Conference Organization

Conference Officials

General Chairs:	Ernesto Damiani Giancarlo Succi	University of Milan, Italy Free University of Bolzano-Bozen, Italy
Track Chairs:	Barbara Russo Scott A. Hissam Björn Lundell	Free University of Bolzano-Bozen, Italy Carnegie Mellon Software Engineering Institute, Pittsburgh, USA University of Skövde, Sweden
Workshop Chair:	Stefano De Panfilis	Engineering Ingeneria Informatica, Italy
Workshop co-Chairs and Web Masters:	Alberto Colombo Fulvio Frati	University of Milan, Italy University of Milan, Italy

Program Committee

Pär J. Ågerfalk <i>Uppsala University, Sweden</i>	Gregory Lopez <i>Thales, France</i>
Olivier Berger <i>Dept. INF - Institut National des Télécommunications, France</i>	Paul Malone <i>Waterford Institute of Technology, Ireland</i>
Gerry Coleman <i>Dundalk Institute of Technology, Ireland</i>	Pentti Marttin <i>Nokia Siemens Networks, Finland</i>
Kieran Conboy <i>National University of Ireland, Galway, Ireland</i>	Catharina Melian <i>Stockholm School of Economics, Sweden</i>
Kevin Crowston <i>Syracuse University, USA</i>	Martin Michlmayr <i>HP, Austria</i>
Jean-Michel Dalle <i>University Pierre-et-Marie-Curie, Paris, France</i>	Sandro Morasca <i>Università degli Studi dell'Insubria, Varese, Italy</i>
Paul A. David <i>Stanford/Oxford University, USA/UK</i>	John Noll <i>Santa Clara University, USA</i>
Stefano De Panfilis <i>Engineering Ingeneria Informatica S.p.a, Italy</i>	Bülent Özel <i>Instanbul Bilgi University, Turkey</i>

Francesco Di Cerbo
Free University of Bolzano-Bozen, Italy

Gabriella Dodero
Free University of Bolzano-Bozen, Italy

Mahmoud Elish
King Fahd University, Saudi Arabia

Joseph Feller
UCC, Ireland

Pat Finnegan
UCC, Ireland

Brian Fitzgerald
Lero, University of Limerick, Ireland

Rishab Aiyer Ghosh
MERIT Netherlands

Jesus Gonzalez-Barahona
Universidad Rey Juan Carlos, Spain

Jeremy Hayes
University College Cork, Ireland

Joseph Kiniry
University College Dublin, Ireland

Stefan Koch
University of Economics and BA, Austria

Derrick Kourie
University of Pretoria, South Africa

Jean Pierre Laisne
ObjectWeb, France

Stéphane Laurière
Mandriva, France

Witold Pedrycz
University of Alberta, Canada

Paolo Pumilia
EST, Italy

Walt Scacchi
University of California, Irvine, USA

Marco Scotto
Free University of Bolzano-Bozen, Italy

Barbara Scozzi
Politecnico di Bari, Italy

Alberto Sillitti
Free University of Bolzano-Bozen, Italy

Gregory Simmons
University of Ballarat, Australia

Sandra A. Slaughter
Georgia Institute of Technology, USA

Diomidis Spinellis
*Athens University of Economics and
Business, Greece*

Frank van der Linden
Philips, The Netherlands

Tony Wasserman
Carnegie-Mellon West, USA

Charles Weinstock
Software Engineering Institute, USA

Hongbo Xu
*South China University Of Technology,
China*

Board of Reviewers

Morkel Theunissen

University of Pretoria, South Africa

List of Contributors

Giovanni Acampora

University of Salerno, IT
gacampora@unisa.it

Claudio Agostino Ardagna

University of Milan, IT
ardagna@dti.unimi.it

Massimo Banzi

Telecom italy, IT
massimo.banzi@telecomitalia.it

Guido Bruno

Telecom italy, IT
guido.bruno@telecomitalia.it

Andrea Capiluppi

University of Lincoln, UK
acapiluppi@lincoln.ac.uk

Davide Dalle Carbonare

Engineering Ingegneria Informatica
S.p.A., IT
davide.dallecarbonare@eng.it

Reidar Conradi

Norwegian University of Science and
Technology, NO
reidar.conradi@idi.ntnu.no

Kevin Crowston

Syracuse University, US
crowston@syr.edu

Ernesto Damiani

University of Milan, IT
damiani@dti.unimi.it

Adriaan de Groot

Sirius Corporation Ltd., UK
groot@kde.org

Paul Adams

Sirius Corporation Ltd., UK
paul.adams@siriusit.co.uk

Marco Antonio Balieiro

Faculdade de Computaçã –
Universidade Federal do ParáBR
ma.balieiro@gmail.com

Andrea Bosin

Università degli Studi di Cagliari, IT
andrea.bosin@dsf.unica.it

Giovanni Caire

Telecom italy, IT
giovanni.caire@telecomitalia.it

Eugenio Capra

Politecnico di Milano, IT
eugenio.capra@polimi.it

Alberto Colombo

University of Milan, IT
colombo@dti.unimi.it

Alain Corbière

Le Mans University, FR
alain.corbiere@univ-lemans.fr

Jean-Michel Dalle

Université Pierre et Marie Curie, FR
jean-michel.dalle@upmc.fr

Vincenzo D'Andrea

University of Trento, IT
vincenzo.dandrea@dit.unitn.it

Stefano De Panfilis

Engineering Ingegneria Informatica, IT
stefano.depanfilis@eng.it

Stefano De Paoli

University of Trento, IT
stefano.depaoli@soc.unitn.it

Cleudson De Souza

Faculdade de Computaçã –
Universidade Federal do ParáBR
cdesouza@ufpa.br

Matthijs den Besten

University of Oxford, UK
matthijs.denbesten@oerc.ox.ac.uk

Nicoletta Dessi

Università degli Studi di Cagliari, IT
desi@unica.it

Leonhard Dobusch

Free University of Berlin, DE
leonhard.dobusch@wiwiss.fu-berlin.de

Nabil El Ioini

Free University of Bolzano-Bozen, IT
Nabil.ElIoini@stud-inf.unibz.it

Joseph Feller

University College Cork, IE
JFeller@afis.ucc.ie

Brian Fitzgerald

University of Limerick, IE
Brian.Fitzgerald@ul.ie

Fulvio Frati

University of Milan, IT
frati@dti.unimi.it

Loris Gaio

Università di Trento, IT
lgaio@cs.unitn.it

Samuel De Sousa Júnior

Faculdade de Computaçã –
Universidade Federal do ParáBR
sfelixjr@gmail.com

Vieri del Bianco

University of Insubria, IT
vieri.delbianco@uninsubria.it

Amit Deshpande

SAP Labs LLC, US
amit@amit-deshpande.com

Francesco Di Cerbo

Free University of Bolzano-Bozen, IT
francesco.dicerbo@unibz.it

Gabriella Dodero

Free University of Bolzano-Bozen, IT
gabriella.dodero@unibz.it

Robert Feldt

Blekinge Institute of Technology, SE
rfd@bth.se

Rafael Fernández

Universidad Politécnica de Madrid, ES

Chiara Francalanci

Politecnico di Milano, IT
francala@elet.polimi.it

Maria Grazia Fugini

Politecnico di Milano, IT
fugini@elet.polimi.it

Les Gasser

University of Illinois, US
gasser@uiuc.edu

Jaco Geldenhuys

Stellenbosch University, ZA
jacogeld@gmail.com

Jesus M Gonzalez-Barahona

Universidad Rey Juan Carlos, ES
jgb@gsync.escet.urjc.es

Georgios Gousios

Athens University of Economics and
Business, GR
gousiosg@aueb.gr

Øyvind Hauge

Norwegian University of Science and
Technology, NO
oyvind.hauge@idi.ntnu.no

Juan José Hierro

Telefónica Investigación y Desarrollo,
ES
jhierro@tid.es

James Howison

Syracuse University, US
jhowison@syr.edu

Juha Järvensivu

Tampere University of Technology, FI
juha.jarvensivu@tut.fi

Tanja Kirves

University of Oulu, FI
tanjakir@mail.student.oulu.fi

Luigi Lavazza

University of Insubria, IT
luigi.lavazza@uninsubria.it

Juho Lindman

Helsinki School of Economics, FI
juho.lindman@hse.fi

Pietro Giovannini

Free University of Bolzano-Bozen, IT
Pietro.Giovannini@stud-inf.unibz.it

Tony Gorschek

Blekinge Institute of Technology, SE
tgo@bth.se

Imed Hammouda

Tampere University of Technology, FI
imed.hammouda@tut.fi

Henrik Hedberg

University of Oulu, FI
henrik.hedberg@oulu.fi

Scott Hissam

Carnegie Mellon
Software Engineering Institute, US
shissam@sei.cmu.edu

Netta Iivari

University of Oulu, FI
netta.iivari@oulu.fi

Yasutaka Kamei

Nara Institute of Science and
Technology, JP
yasuta-k@is.naist.jp

Stefan Koch

Vienna University of Economics and
BA, AT
stefan.koch@wu-wien.ac.at

Diego Liberati

IEIIT CNR c/o Politecnico di Milano, IT
liberati@elet.polimi.it

Vincenzo Loia

University of Salerno, IT
loia@unisa.it

Crista Lopes

University of California, US
lopes@ics.uci.edu

Max Loubser

Oxford Internet Institute, UK
max.loubser@magd.ox.ac.uk

Hiroataka Maeshima

Nara Institute of Science and
Technology, JP
hirotaka-m@is.naist.jp

Andrés Martínez

Universidad Rey Juan Carlos, ES
aleonar@gsyc.escet.urjc.es

Pentti Marttiin

Nokia Siemens Networks, FI
pentti.marttiin@nsn.com

Ken-ichi Matsumoto

Nara Institute of Science and
Technology, JP
matumoto@is.naist.jp

Aurelien Mayoue

TELECOM & Management SudParis,
FR
aurelien.mayoue@int-edu.eu

Francesco Merlo

Politecnico di Milano, IT
merlo@elet.polimi.it

Sandro Morasca

University of Insubria, IT
sandro.morasca@uninsubria.it

Masao Ohira

Nara Institute of Science and
Technology, JP
masao@is.naist.jp

Dario Lorenzi

Politecnico di Milano, IT
dario.lorenzi@polimi.it

Björn Lundell

University of Skövde, SE
bjorn.lundell@his.se

Magnus Mähring

Stockholm School of Economics, SE
magnus.mahring@hhs.se

Juan Martinez-Romo

E.T.S.I. Informatica (UNED), ES
juaner@lsi.uned.es

Héla Masmoudi

Université Pierre et Marie Curie, FR
masmoudi_hela@yahoo.fr

Shinsuke Matsumoto

Nara Institute of Science and
Technology, JP
shinsuke-m@is.naist.jp

Catharina Melian

Stockholm School of Economics, SE
catharina.melian@hhs.se

Tommi Mikkonen

Tampere University of Technology, FI
tommi.mikkonen@tut.fi

John Noll

Santa Clara University, US
jhnoll@gmail.com

Yoji Onishi

Nara Institute of Science and
Technology, JP
yoji-o@is.naist.jp

Heikki Orsila

Tampere University of Technology, FI
heikki.orsila@tut.fi

Barbara Pes

Università degli Studi di Cagliari, IT
pes@unica.it

Dijana Petrovska-Delacretaz

TELECOM & Management SudParis,
FR
dijana.petrovska@int-edu.eu

Pascal Ravesteijn

University of Applied Science Utrecht,
NL
pascal.ravesteijn@hu.nl

Dirk Riehle

SAP Labs LLC, US
dirk@riehle.org

Alessandro Rossi

Università Trento, IT
arossi@cs.unitn.it

Matti Rossi

Helsinki School of Economics, FI
matti.rossi@hse.fi

Barbara Russo

Free University of Bolzano-Bozen, IT
Barbara.Russo@unibz.it

Walt Scacchi

University of California, US
wscacchi@uci.edu

Alberto Sillitti

University of Bolzano, IT
alberto.sillitti@unibz.it

Miguel A. Ortuño-Perez

Universidad Rey Juan Carlos, ES
mortuno@gsyc.escet.urjc.es

Etiel Petrinja

University of Bolzano, IT
etiel.petrinja@unibz.it

Alvaro Polo

Universidad Rey Juan Carlos, ES
apoloval@gsyc.escet.urjc.es

Marcos Reyes

Telefónica Investigación y Desarrollo,
ES
mru@tid.es

Gregorio Robles

Universidad Rey Juan Carlos, ES
grex@gsyc.escet.urjc.es

Cristina Rossi

Politecnico di Milano, IT
cristina.l.rossi@polimi.it

Anna Ruokonen

Tampere University of Technology, FI
anna.ruokonen@tut.fi

Ioannis Samoladas

Aristotle University of Thessaloniki, GR
ioansam@csd.auth.gr

Nico Schellingerhout

Philips Medical Systems, NL
nico.schellingerhout@philips.com

Gilbert Silvius

University of Applied Science Utrecht,
NL
gilbert.silvius@hu.nl

Susan Elliott Sim

University of California, US
ses@ics.uci.edu

Javier Soriano

Universidad Politénica de Madrid, ES

Diomidis Spinellis

Athens University of Economics and
Business, GR
dds@aueb.gr

Volker Stix

Vienna University of Economics and
BA, AT
volker.stix@wu-wien.ac.at

Davide Taibi

University of Insubria, IT
davide.taibi@uninsubria.it

Maurizio Teli

University of Trento, IT
maurizio.teli@soc.unitn.it

Medha Umarji

University of Maryland Baltimore
County, US
medha1@umbc.edu

Andrea Wiggins

Syracuse University, US
awiggins@syr.edu

Carl-Fredrik Sørensen

Norwegian University of Science and
Technology, NO
carl.fredrik.sorensen@idi.ntnu.no

Sulayman K. Sowe

Aristotle University of Thessaloniki, GR
sksowe@csd.auth.gr

Ioannis Stamelos

Aristotle University of Thessaloniki, GR
stamelos@csd.auth.gr

Giancarlo Succi

Free University of Bolzano-Bozen, IT
giancarlo.succi@unibz.it

Romaric Tchokpon

Institut de Mathématiques et de Sciences
Physiques, BJ
ricotchfr@yahoo.fr

Richard Torkar

Blekinge Institute of Technology, SE
rto@bth.se

Anthony I. Wasserman

Carnegie Mellon West, US
tonyw@west.cmu.edu

A Framework for Evaluating Managerial Styles in Open Source Projects

Eugenio Capra¹ and Anthony I. Wasserman²

¹ Department of Electronics and Information, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy, eugenio.capra@polimi.it

² Center for Open Source Investigation, Carnegie Mellon West, Mountain View, CA 94035 USA, tonyw@west.cmu.edu

Abstract. This paper presents the Software Project Governance Framework (SPGF) for characterizing management of software projects, based on mechanisms used for communication and collaboration, the organizational structure of projects, and testing and quality assurance procedures. The framework was developed and validated from interviews and surveys with leaders of more than 70 commercial and community-based software projects, including both closed and open source projects.

Keywords: Management of OSS development; Working practices; Industry contribution; Community contribution

1 Introduction

Open source software and accompanying “open” development practices have had a major impact on the software industry. “Open” software development processes involve new managerial styles, governance and social models, working practices and communication techniques (cf. [1], [4], [5], [9], and [10]).

Open source products fall into two major categories, which we term “community” and “commercial”. Community Open Source projects are led by a community of developers or stakeholders and are distributed under an approved open source license, e.g., GPL, BSD, or Apache. Companies or institutions may have a significant role in the governance of the project, and may contribute many of the resources needed for the ongoing development of the project, but there are few, if any, limitations on who may participate in the various aspects of the project. Development is done “in the open” so that anyone may have complete, no-cost access to the current state of the project. These projects, such as those sponsored by the Apache Software Foundation, have established policies for granting “commit rights” that allow individuals to modify the code base.

Commercial Open Source projects are led by a company, which has usually developed most or all of the code, and then sells subscriptions and services for the developed product. Commercial Open Source applications are very often distributed with a dual license scheme, one offering unrestricted use of the software (community version) and one intended for commercial use of the software. In some cases, the two

Please use the following format when citing this chapter:

Capra, E. and Wasserman, A.I., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 1–14.

versions of the software differ, with the commercial version including features that are not present in the unrestricted version. In that situation, the commercial version of the software typically includes some closed source code. Also, the license for the community version may not be an “approved”, i.e., an OSI-listed, open source license.

These approaches are beginning to blend with traditional closed-source software development. Numerous companies offer both closed and open source products, and also participate in non-commercial open source projects. In many cases, companies have completely different policies for each type of project.

While open source is technically a licensing model, its impact on software development processes goes well beyond licensing. The open source phenomenon has had a global impact on the way organizations and individuals create, distribute, and use software, and has challenged the conventional wisdom of the software engineering and software business communities. For this reason, we have focused on managerial and governance approaches to open source projects with the primary goal of creating a framework to characterize these different managerial styles and to evaluate the “openness” of a software project (as opposed to the software itself). The resulting framework allows potential users of the project to identify well-managed projects and allows potential contributors to see if there is a good opportunity to participate in the project.

Governance theory has been applied to software development in a number of different approaches [6], and is defined as the complex process that is responsible for the control of project scope, progress, and continuous commitment of developers [8]. It would be very difficult to elaborate a framework able to encompass all the possible governance dimensions of a software project. We propose a governance framework that allows one to position a software project along the continuum between “fully open” and “fully closed” approach.

This paper is organized as follows: Section 2 gives a brief overview of the study we conducted to define the framework. Section 3 presents the framework with some quantitative and qualitative metrics to evaluate software projects along several dimensions. Section 4 describes some existing open source projects and discusses how they can be classified according to our framework as an example. Section 5 briefly discusses how the framework was applied to our sample. Finally, Section 6 gives preliminary conclusions and topics for further study.

2 Methodology

Our framework aims at positioning a software project along the continuum between fully open and fully closed governance practices. It was developed through preliminary empirical analysis based on individual face-to-face interviews with 25 project managers of major software projects along the continuum, including

traditional closed source development projects (packaged software and software as a service), Commercial Open Source and Community Open Source projects. Project managers were asked which governance dimensions were most significant to measure the degree of openness of a software project. We identified four fundamental governance dimensions: contribution, project leadership, working practices, and testing. These dimensions were chosen since they were widely cited by the project managers, and since they had the highest ranking of all of the cited dimensions.

Subsequently, we refined and validated our framework through a continuing study of more than 70 software projects. We included projects from SourceForge.org, Apache.org, Tigris and Java.net that met the following criteria:

- Mature status (according to the classification provided by the repositories, when available, or to common sense for major projects);
- At least 2 administrators (committers);
- At least 2 developers or contributors.

We have focused on large and well-known projects, rather than those developed by small teams, because these projects had developed and evolved their managerial and governance approaches. Our goal was to identify the dimensions that best illustrate the continuum between open and closed governance approaches. Our research approach has been informal, aimed at identifying the key dimensions and differentiators among projects of varying age, size, diffusion and domain. Table 1 describes how these parameters vary across the sample.

Table 1 – Description of age, size and diffusion of the projects analyzed.

Variable	Minimum value	Average value	Maximum value
Age [year]	<1	8	30
Size [core developers]	10	100	1,000
Size [kSLOC]	40	1,000	6,000
Diffusion [downloads or users]	2	25,000	200,000

Data was collected through interviews with and surveys of key project personnel, namely community managers, QA managers, VPs of engineering, committers, and project leaders, with follow-up calls made as needed for clarification and consistency.

The interviews focused on the following topics:

- *Governance and organization*: Is the project more similar to a “benevolent dictatorship” or to a democracy? Is it self-organizing or centrally controlled?

What is the role of the internal community versus the external community? How many developers are paid?

- *Work practices and tools*: How is the right to commit code granted? How is code reviewed? How important is automated testing? How many management tasks and non-code-developing tasks are shared in the community?
- *Communication and social culture*: Which tools (cvs, bug tracking, IRC, wiki, etc.) are used? How frequent is face-to-face communication? How open are discussions? How is consensus reached?
- *Comparison between open and traditional projects*: How do closed and open projects differ in management practice? What are the relative advantages of open source development compared with traditional closed development?

The results of these interviews formed the basis for our framework, which we call the Software Project Governance Framework (SPGF). The methodology we adopted to formalize the evidence we gathered is based on three major steps. First, we identified and characterized two hypothetical projects representing the two extremes of the spectrum, i.e. a completely traditional closed software project and a completely open source community-based project (see also [3]). Second, we defined dimensions along which these projects can be evaluated, eventually selecting four dimensions that gave the most accurate picture of governance. Third, we scored each dimension of each project from 1 to 4, where 1 indicates a closed-style approach and 4 signifies an open-style approach. We show the detail of the scoring for each dimension below. Note that neither licensing nor the distribution model are part of the framework.

The SPGF framework provides a qualitative assessment of the degree of openness and, accordingly, scales are ordinal. Assessing governance by means of ratio variables not only is difficult, but may also be misleading [5], [11]. The SPGF framework is intended for comparing projects rather than providing absolute assessments. Moreover, the output of our framework may be employed within quantitative methodology according to the approach discussed by Briand et al. in [1].

We would also note that the interviews covered a wide range of topics, and our dimensions have been extracted as the most important factors to distinguish different approaches to project governance. Some of the interviews ranged beyond the specific issues of the framework and helped us to validate the overall approach.

3 The Framework

This section presents our framework. First, we characterize the properties of a completely closed and a completely open project. Then we describe the four dimensions at the base of the framework and provide a graphical representation methodology.

3.1 A traditional closed source project

A “traditional” software project is led by a company or an organization which strictly controls the development process. The proprietary code is closed and is developed by paid staff, possibly including contractors or outsourced teams. Most projects have a well-defined organizational structure following a development process aimed at producing a high quality product (or service) on a predictable schedule. Members of the team “meet” regularly, and report their progress through their organization’s management structure.

Many companies have user groups, advisory boards, forums, and other ways for users to interact with the development team, but the final decisions are all made by the company, which has responsibility for all of the code and documentation. In general, the development team has its own communication mechanism, which is not open to outsiders.

The company does most of its own testing and fixes problems even before releasing a beta version to users. Many make their beta versions available to a broad community of users, providing mechanisms for reporting issues and problems in functionality, performance, installation, usability, stability, and/or security.

3.2 A completely open software project

At the opposite end of the spectrum are the thousands of Community Open Source projects, each with its own community, open to anyone who is interested in the project. The work is done entirely in the open, and is typically hosted in such repositories as SourceForge, Tigris.org, Apache Software Foundation, and Java.net. The software can be acquired and used by anyone, subject to the terms of the project’s license agreement.

In an open source project, a project lead (or leadership group) is responsible for overall project management, such as determining when a version of the software is ready (stable version), selecting the license to be used with the software release, and deciding who can have “commit rights” to the code.

Some projects are very informal, without formal organization and governance bodies. Decisions are usually made by informally discussing issues within forums, mailing lists or IRC channels. Some communities may have a voting mechanism for resolving issues.

Project participation is open to all, independent of organizational affiliation. Many projects include both volunteers, who have another job and work on the project in their spare time, as well as people who are paid by companies to work on the project.

Since project participation is often a volunteer activity, the project leadership cannot easily compel someone to work on a specific task or to adhere to a schedule, as is the case in a commercial software project. Participants in these community-

based projects rarely meet in person. Instead, they communicate by mechanisms such as forums, mailing lists, IRC channels, instant messaging systems, wikis, blogs, online shared task lists or similar devices. Each community relies on one or more of these tools according to its tradition and habits.

A Community Open Source project doesn't have formal testing or quality assurance processes, but instead relies upon individual developers to test their own code, and for community members to test the software and post issues (and possibly fixes) using the project's issue tracking system. Well-managed projects respond quickly to posted bugs, relying on individual committers to make any needed changes or enhancements to their code. While commercial projects control the number of releases and offer customer support for those releases, no comparable support mechanism is in place for community-based open source projects.

3.3 Dimensions of the SPGF

Using these typical approaches for project management, we defined the following dimensions along which software projects can be evaluated.

3.3.1 Contributions

This dimension measures the relative amount of voluntary code development. Most Commercial Open Source companies resemble proprietary software companies in their reliance on paid development.

In a community-based open source project, code is usually developed on a voluntary basis. However, contributors may be employed or hired by a company or an organization that wants to lead the project or to accomplish specific tasks (e.g., to implement a new feature or to fix a specific bug).

A significant difference between hired and voluntary developers is that the former have to follow the guidelines and deadlines imposed by their employers, whereas the latter are really free to work according to their will and inclination.

Table 2 provides a quantitative metric for this dimension.

Please note that we use the term *hired developer* rather than *employee* as a way to distinguish volunteers from people who receive regular compensation for their contributions to a software project. Whoever is the employer and whatever the form of contract, a person who is paid to develop an application will behave differently from a person who writes code in his spare time just for personal satisfaction. Some companies pay a nominal "bounty" to individuals for small contributions; we do not consider them to be hired developers.

We used 80% as a threshold since the percentage of code committed by volunteers on commercial projects is typically below 10%. In community-based projects, less than 50% of the code is developed by hired employees.

Table 2 – Evaluation of *contributions* dimension.

Value	Description
1	100% of the code is developed by hired developers
2	>80% of the code is developed by hired developers
3	>50% of the code is developed by hired developers
4	Most of the code is developed by volunteers

3.3.2 Project leadership

This dimension indicates the degree to which the leadership of a project is hierarchical. Commercial Open Source projects are led by a company, which usually defines a roadmap and sets schedules. Companies might also play a significant role in guiding and managing community projects. Some Community Open Source projects are indirectly governed by a predominant company, which defines the roadmap of the project and leverages the community to reach its goals. Communities may be led not only by a company, but also by a foundation or by an independent committee. Some projects are managed by a “benevolent dictator”: participation and discussion are fostered, but final decisions are made by the project leader or an entrusted committee. The Linux Kernel project is an example this style of governance. On the other hand, fully open communities often lack a formal organization. Decisions are made by voting or by governance bodies which are directly elected by active contributors. Less formal communities adopt the *lazy consensus* approach, i.e., issues are discussed within forums and mailing lists and decisions are made when nobody has anything more to add.

It is very difficult to provide a quantitative metric to evaluate this dimension. For the cases we analyzed, we developed a qualitative scale, shown in Table 3.

Table 3 – Evaluation of *Project leadership* dimension.

Value	Description
1	Roadmap and development process are led by one company or organization which has a predominant leadership role, makes decisions and sets schedules.
2	Roadmap and development process are led by one company or organization. However, free discussion and participation to the governance of the project is fostered.
3	The community is ruled by some formal rules and principles. Decisions are made mainly by voting or by governance bodies directly elected by contributors.
4	The community completely lacks a formal organization and governance bodies. Decisions are made by informally discussing issues.

3.3.3 Working practices

This dimension indicates the degree to which the working and communication practices of a project are geographically distributed and virtual.

Proprietary software projects and many Commercial Open Source projects rely primarily on a closed community working for a single employer, often in close physical proximity. A Community Open Source project, by contrast, often has a geographically dispersed membership. With little funding to support physical meetings of the project team, these projects rely heavily on collaborative tools. Note that such tools may also be used by those in close proximity to each other.

Table 4 presents a qualitative scale for this dimension.

Table 4 – Evaluation of *working practices* dimension.

Value	Description
1	Developers work on the same site, communicate in traditional ways and have regular physical meetings.
2	Most developers work on the same site and have regular physical meetings, with some remote participants
3	The community is dispersed and most developers are remote. Some subsets of developers, however, work at the same location and meet regularly.
4	The community is widely dispersed and all the developers communicate through virtual tools. Physical meetings are totally absent or very rare (1-2 per year).

3.3.4 Testing

This dimension aims at describing the testing process, as well as the presence and role of a Quality Assurance department (or resources) within the project.

As noted in Section 3.1, commercial software development organizations typically have Quality Assurance departments that define formal test processes and are responsible for the quality of the application. A Quality Assurance department also defines quality standards, including those for contributions submitted for inclusion in the code base.

By contrast, Community Open Source projects rely on their own developers and their user community for testing, with relatively few formal processes or tools. In general, open source projects tend not to have specific QA roles, even though some open source projects have very strict pair reviewing rules that determine when new code or patches can be committed to the code base.

Table 5 provides a qualitative scale for this dimension. Please note that by “internally” we also mean testing done by the committers or the core developers of a project. The word “community” in this context refers to users or casual contributors.

Table 5 – Evaluation of *testing* dimension.

Value	Description
1	All the testing is controlled internally by specific QA roles. New versions of the application are released only after being thoroughly tested.
2	Most testing (>50%) is performed internally before new versions of the application are released. The user community is leveraged as a broader testing platform, for example by releasing beta versions and then collecting feedback and bug notifications.
3	Some testing (<=50%) is performed internally, but most of it is left to the community of users.
4	Testing is completely left to the community of users.

3.3.5 Graphical representation

We use a diamond graph to show where projects fall on the spectrum for each dimension. Figure 1 shows the extreme cases of a traditional closed source and a completely open software projects.

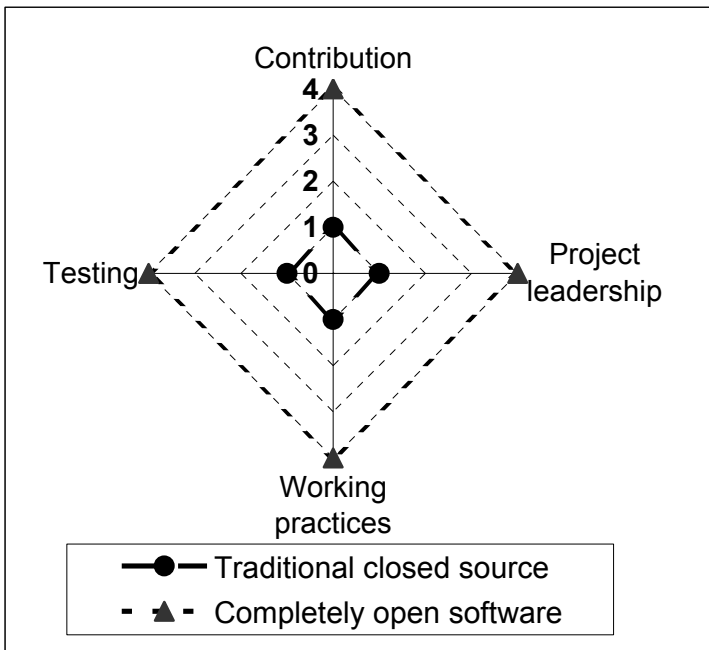


Fig. 1 – Graphical representation of project management dimensions.

4 Case Studies

In this section we provide some examples on how the SPGF may be applied to real projects. We apply the SPGF to three open source projects: OpenOffice.org, MySQL and SugarCRM. We chose these since they are well known applications, and show differences among the dimensions of the framework.

Figure 2 presents a graphical representation of the positioning of these projects. A first glance at the picture shows that OpenOffice.org and MySQL are closer to the completely open source approach, while SugarCRM is closer to closed software projects.

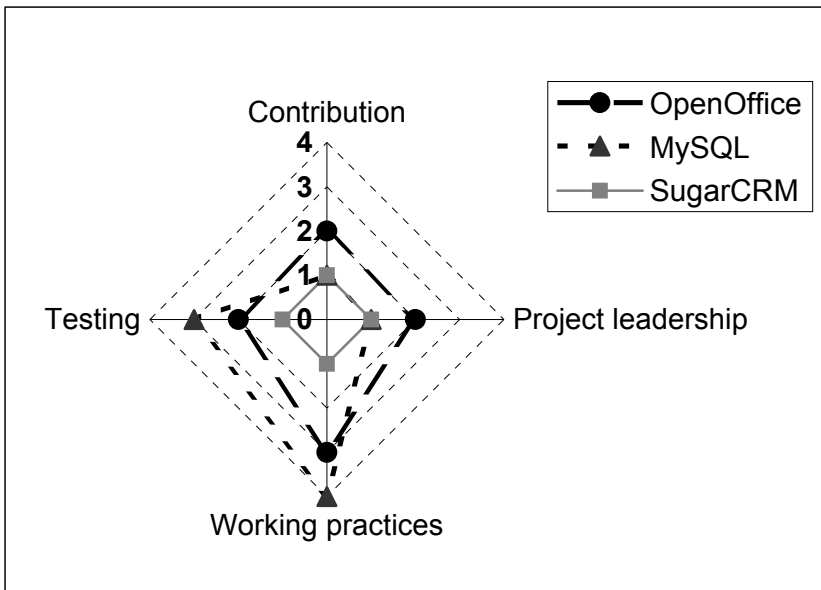


Fig. 2 – Graphical representation of the assessment of a project according to the framework.

4.1 OpenOffice

OpenOffice (OpenOffice.org) is a widely used open source office suite with more than 100 million downloads.

OpenOffice is quite a monolithic project. Although everybody can contribute to the project and can earn commit right, Sun Microsystems and IBM have historically contributed almost 90% of the code, paying more than 90 developers for their work. Other companies, such as RedHat and Novell, also contribute to the code. This accounts for the score 2 on *contribution* dimension.

The community has a very structured governance model, based on a Community Council and an Engineering Steering Committee. The Community Council is constituted by members of the community but is deeply influenced by Sun and IBM. The project has a clear and shared roadmap, which probably could not exist without a corporate structure in the background. All these factors lead to score 2 on the *project leadership* dimension.

Communication within the community mainly takes place on mailing lists and on IRC channels. However, most of Sun's developers work in Hamburg and meet daily. As a result, issues are often discussed in person and then conclusions are posted on mailing lists, so that remote community members can be informed. Moreover, occasional cross-corporation meetings are held several times a year. Consequently, *working practices* score 3.

OpenOffice began as StarOffice with Sun, and the QA team that worked on StarOffice now works on OpenOffice. There are currently about 550 QA members with *canconfirm* privilege, i.e. the ability to approve some feature before it is issued. In this particular aspect, OpenOffice is very similar to a traditional software house. Testing is also managed in a very structured way. Specific test suites have been written, integration and system testing are carried out regularly, daily smoke tests, regression testing and code coverage tools are adopted on a regular basis. Every developer is responsible for testing his code, but pair review is applied, too, and the QA team has to confirm the validity of new code. Feedback and bug notifications from users are also accepted and encouraged. This behavior accounts for score 2 on *testing* dimension.

4.2 MySQL

MySQL (www.MySQL.com) is distributed by MySQL, AB (now part of Sun Microsystems). Even though the code is open, it is mainly developed (99%) by employees of the company. The community is invited to submit new code, which is reviewed according to strict and documented internal standards before it is accepted. However, this is quite rare, given the size and complexity of the code base. This accounts for the score 1 on *contribution* dimension.

MySQL (the company) controls governance of the project. The corporate culture is very open to discussion, which is fostered by means of online communication tools, such as blogs, wikis, and forums, but MySQL, as a traditional software house, makes the final decisions. Thus, we assign a score of 1 to *project leadership* dimension.

The real value of the community is mainly to create a broad marketing platform and to provide extensive testing that augments the internal MySQL QA department. Functional tests and cross-platform tests are usually done by the internal development team, then QA tests the alpha version using their own scripts. Once the code is released, more than 50% of testing is left to the community, which also

performs most of the integration tests. This combination of internal QA and external testing explains the scores 3 on the *testing* dimension.

Although MySQL is managed as a traditional company, many of its working practices resemble those of community projects. Developers are located in 26 countries around the world, and work from home, meeting only once or twice a year. They mainly communicate through asynchronous tools, such as highly specific internal IRC channels, shared task lists and e-mails, to overcome time zone differences. Telephone conference calls and video chats are also organized, but they are always combined with e-mails or forum posts. This accounts for the score of 4 on the *working practices* dimension.

4.3 *SugarCRM*

SugarCRM (www.SugarCRM.com) is another Commercial Open Source project. Similarly to MySQL, most of the code is open, but it is developed by internal employees only. The core application is centrally controlled by the company, while the community is involved in the creation of new projects, such as extensions and plug-ins, which are hosted on the SugarForge website. Consequently, the score for *contribution* and *project leadership* dimensions is the same as MySQL.

On the other hand, most of the developers work in the same location and have regular meetings. Forums and mailing lists are used, but by external community members rather than internal developers. VoIP phone conferences are frequent, but this happens even in very traditional closed source projects. Consequently, it scores 1 on the *working practices* dimension.

Most quality assurance and testing is performed by the internal QA department, which is also responsible for bug fixing. This accounts for score 1 on the *testing* dimension.

SugarCRM governance and managerial styles are actually very similar to those of a traditional closed software project, with the only exceptions that most of the code is open and that external people can contribute code.

5 Application of the Framework to the Sample

After defining the framework, we applied it to our sample of Community Open Source projects. Table 6 shows the distribution of the scoring of the projects in the sample along the four dimensions of the framework.

Table 6 – Distribution of SPGF scores across the sample.

Dimension x	x<3	3<=x<4	x=4
Contribution	43%	10%	47%
Project Leadership	30%	48%	22%
Working Practice	7%	37%	57%
Testing	33%	44%	15%

Most of the communities have some kind of organization and governance bodies, which control new contributions and part of the testing. In particular, the survey showed that approximately 50% of the code of the applications in the sample is developed by hired developers and that physical meeting are held in 35% of projects.

6 Conclusions and Future Work

The Software Project Governance Framework provides a consistent way to analyze projects based on their governance and managerial styles. The central idea behind the framework is that open source has a deep impact on the governance of a software project and, consequently, may impact its quality and costs. The empirical analyses we conducted allowed us to study and embrace a wide range of different software projects. The SPGF provides a structured methodology to analyze managerial and governance models, and to categorize these projects according to the dimensions that are regarded as the most significant by the project leaders. We think that the SPGF may enable a deeper comprehension of software projects and may be useful to a wide range of users.

First, it may be used by researchers to quickly assess and cluster projects. This allows one to select a homogenous sample of projects from a governance point of view before performing further surveys and analysis. We are working on a research project that seeks correlations between the SPGF dimensions and quality of design and development effort of a software project. Second, this framework is valuable to end users seeking information about the structure of various open source projects. For example, a company which is evaluating the adoption of an open source application may want to know and classify the governance approach behind the development of that application. Third, this framework may be used as a reference by developers and project leaders who want to position their products among the different typologies of open source projects and clearly present their managerial style to the public.

In the future, we are planning to further validate and potentially extend this framework. We will expand our sample through additional interviews and surveys, and also seek correlations between these dimensions and project success.

Acknowledgments

We are grateful to the project managers who provided data to us. For the projects identified in this paper, we specifically acknowledge the participation of Louis Suarez-Potts (OpenOffice), Kaj Arnö and Omer BarNir (MySQL), and Jacob Taylor (SugarCRM). We also thank Professor Chiara Francalanci and Francesco Merlo (Politecnico di Milano) for their support and advice.

References

- [1] L.C. Briand, K. El Emam, and S. Morasca, “On the application of measurement theory in software engineering”, *Journal of Empirical Software Engineering*, vol. 1, no. 1, pp. 61-88, 1996
- [2] B. Fitzgerald, “The Transformation of Open Source Software”, *MIS Quarterly*, vol. 30, no. 3, 2006.
- [3] K. Fogel, “Producing Open Source Software”, O’Reilly, Sebastopol (CA), 2006.
- [4] G. Goth, “Open Source Business Models: Ready for Prime Time”, *IEEE Software*, Nov/Dec 2005, pp. 99-100.
- [5] M. Griffiths. (2006, Oct. 5). Most software development metrics are misleading and counterproductive [Online]. *Agile Journal*, Available: <http://www.agilejournal.com/content/view/107>
- [6] L.J. Kirsch, “The management of complex tasks in organizations: controlling the systems development process”, *Organization Science*, vol. 7, no. 1, pp. 1-21, 1996.
- [7] A. MacCormack, J. Rusnak, and C.Y. Baldwin, “Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code”, *Management Science* (forthcoming).
- [8] P.S. Renz, *Project governance: implementing corporate governance and business ethics in nonprofit organizations*, Heidelberg, Physica-Verl, 2007.
- [9] S. Slaughter, J. Roberts, and I. Hann, “Communication Networks in an Open Source Software Project”, *Proc. of 2nd Conference on Open Source Systems*, Italy, Jun 2006.
- [10] S. Slaughter, J. Roberts, and I. Hann, “Motivations, Participation and Performance in Open Source Software Development”, *Management Science*, (forthcoming).
- [11] J. Sonnenfeld, “Good governance and the misleading myths of bad metrics”, *Academy of Management Executives*, vol. 18, no. 1, pp. 108-113, 2004.

Forging A Community – Not: Experiences On Establishing An Open Source Project

Juha Järvensivu¹ and Tommi Mikkonen¹

¹Institute of Software Systems, Tampere University of Technology
Korkeakoulunkatu 1, FI-33720 Tampere, Finland
{juha.jarvensivu, tommi.mikkonen}@tut.fi

Abstract. Open source has recently become a practical and advocated fashion to develop, integrate, and license software. As a consequence, open source communities that commonly perform the development work are becoming important in the practice of software engineering. A community that is lively can often produce high-quality systems that continuously grow in terms of features, whereas communities that do not gain interest will inevitably perish. Despite their newly established central role, creation, organization, and management of such communities have not yet been widely studied from the viewpoint of software engineering practices. In this paper, we discuss experiences gained in the scope of Laika, an open source project established to develop an integrated software development environment for developing applications that run in Linux based mobile devices.

Keywords: Software engineering, open source community establishment

1 Introduction

Open source development has recently become a practical fashion to develop different types of software systems. This also affects commercial systems, where open source code can be used in tools, platforms, and general-purpose libraries, for instance. Also commercial systems that are largely based on open source components exist, as contrary to the common misbelief these are not contradicting concepts. A good example of such a system is Nokia 770 Internet Tablet, which is based on popular open source components.

Open source software development takes place in communities that developers participate to compose, maintain and further develop associated software. Then, the fashion such communities operate and how they are composed of becomes an important aspect for the future of the system as well as for its users.

Some the differences of open source and proprietary software have already been addressed (e.g. [6], [14]). However, there are little practical research results on several key questions related to communities as software developing entities. Such include

Please use the following format when citing this chapter:

Järvensivu, G. and Mikkonen, T., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 15–27.

how to organize a community in a meaningful and long-lasting fashion,
how to identify a community that is the most promising from others developing similar systems,

how to ensure that the results that a community has produced will be maintained and remain readily available in the future, even if it seems that like software development in general, also communities can be very person-dependent [1].

Moreover, life cycles of open source systems and communities maintaining them are rarely documented explicitly, although a lot of practical knowledge definitely exists among practitioners and the history of high-profile communities is definitely widely and well known.

The first step towards understanding community driven software development practices is to document and share experiences from establishing and maintaining an open source community. In this paper, we provide an overview to project Laika, which has developed an integrated development environment (IDE) based on Eclipse for Nokia Internet Tablet type of devices (N770, N800) running Linux. In the beginning, we were counting on the interest among developers working with the platform, as facilities that readily existed in this environment were not too sophisticated. At first, things looked promising. We were able to compose a fully functional version of the system, and gained a lot of outside interest and publicity. At present, we however face difficulties in supporting the new versions of both the platform and other programs that act as components in our system. The community has simply not been attractive enough for other developers to join, something we took for granted for long time during the project.

The rest of this paper presents our experiences structured as follows. Section 2 discusses our motivation and the target environment we had in mind when we started the project. Section 3 discusses the history of the project, and describes the phases that we have been able to identify in the progress. Section 4 forms the core of the paper by describing our experiences on the creation and maintenance of an open source community. Section 5 draws some final conclusions.

2 Target Environment

The open source project addressed in this paper, Laika, aims at the creation of an integrated development environment (IDE) for developing applications for embedded Linux devices. The particular devices we had in mind, Nokia N770 and N800 Internet Tablets, are based on the Maemo platform [15], and provide Internet browsing capabilities with a relatively large high-resolution touch screen.

The goal of our project was to integrate the work of several open source development projects into a single tool that is sophisticated enough for industry scale software development. When doing so, we planned to implement the necessary glue code ourselves, but rely on the effort of other communities for everything else.

In the beginning, we could readily find several open source projects whose output was valuable for Laika. Most importantly, we decided to build our project on

Maemo [15], Scratchbox [18], and Eclipse [4], which will be addressed in more detail in the following. In addition, we have used other components as well, although their role can be considered less important than that of the above projects. All open source components that are currently included in Laika are listed in Table 1.

Table 1. Component communities of Laika

Component	Description
CDT	C/C++ development environment for Eclipse, which served as a starting point for our development effort.
Eclipse	Vendor-neutral open development platform, whose plugin Laika is.
Glade/Gazpacho	Graphical user interface builder for GTK+.
Maemo	Development platform to create applications for the Nokia 770.
PyDev	Python development environment for the Eclipse platform.
Scratchbox	A cross-compilation toolkit used by the Maemo platform.

Maemo. Maemo, which acts as the software platform for Nokia’s N770 and N800 Internet Tablet, is composed of a set of popular open source software components that are widely deployed in today’s leading desktop Linux distributions [15]. Maemo consists of precompiled Linux kernel, platform libraries, and Hildon user interface framework. The Hildon UI framework in turn is based on GTK+, and as a whole the platform is binary compatible with GTK+ binaries [7]. The structure of the system is illustrated in Fig. 1, and illustrated components are introduced in more detail in Table 2.

Scratchbox. Currently available mobile devices based on Maemo run on ARM processor, whereas most developers prefer using an Intel-based computer like a PC as their development environment. The creation of ARM binaries with such development setup requires cross-compilation, where a host computer is used to compile software for a target that uses a different instruction set. In our case, we have performed cross-compilation using a PC running Scratchbox, which is a toolkit designed to ease embedded Linux application development [18]. The target platform is ARM, the hardware environment of our devices.

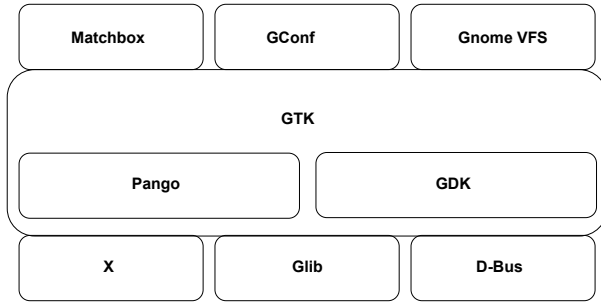


Fig. 1. Maemo's main software components

Table 2. Maemo's main software components

Component	Description
Matchbox	A light-weight window manager that is responsible for managing X11 client window geometry, stacking, and decorations in our target device.
GConf	A generic system for storing application preferences.
Gnome VFS	Gnome virtual file system, provides an abstraction layer for accessing files in both local file systems and the web.
GTK+	The Gimp toolkit, a toolkit for creating windowing applications. The kit also provides an extensive set of readily available widgets.
Pango	A library for text layout, rendering, and internationalization.
GDK	Gimp drawing kit, a graphics library that wraps routines providing low-level access to a display.
X	X Window System, a networking and display protocol for building windowing systems.
Glib	A low-level library that contains commonly needed generic routines for wrappers and runtime routines including event handling, dynamic loading, and threading.
D-Bus	Light-weight inter-process communication system.

Eclipse. The Eclipse platform is a vendor-neutral open development platform that provides tools for managing workspaces and building, debugging, and launching applications for building integrated development environments (IDEs) [4]. In essence, the Eclipse platform is a framework and toolkit that provides a foundation for running third-party development tools. The basic mechanism of extensibility in Eclipse is adding new plugins, which can also add new processing elements to plugins that already exist. In accordance to this architecture, the Laika plugin is based on the C/C++ development tools (CDT) plugin [3], which in turn is a subproject of the Eclipse community [4].

3 Short History of Project Laika

In this section, we describe the history of project Laika. The goal is to give an overview to the main events that the project has experienced, and how we have ended up in the current situation.

3.1 Becoming of Age

The origins of Laika are in meetings between Institute of Software Systems at Tampere University of Technology (TUT) and Nokia Multimedia (Nokia) during the Fall 2004, when the first Maemo based device was nearing completion. A common goal was set by both parties to aim at the development of an industry-strength integrated development environment that would encapsulate facilities necessary for Maemo development. The plan was that in the first phase, we would only implement minimal functionality, which could later be extended. The extensions would be made by either the original developers that were responsible for the first release, or by other parties that we would be able to attract to participate in the community. A related plan was that the experiment would also give the university first-hand information on how open source communities work.

As already documented in [9], the development was initiated by establishing a co-located team of students working as summer-time trainees within the premises provided by the university. Moreover, university already had existing server infrastructure that could be used for hosting the community and enabling download of Laika software. Thus, the initial investment was small, and the actual work could be started within a short period of time from the actual decision.

At the university, the project was supervised by a graduate student and a professor who had participated in meetings with Nokia. As a practical starting point, some code was provided to the community by Nokia Research Center, which had already investigated an opportunity to develop such an IDE [19]. This code then formed the baseline for further development that was carried out by the Laika community.

The first actual release of Laika was made towards the end of the summer of 2005, with the majority, if not all, the work performed by the summer trainees. At this point, the system simply comprised Eclipse, CDT, and Scratchbox, as well as some compilation and debugging capabilities that we felt were important for the project. The system is illustrated in Fig. 2.

The 1st release proved that there indeed was a need for an IDE for Maemo, and that Laika was really something that people found useful, at least based on the number of download and the feedback received from people installing and using the system. A total of 603 downloads were made during the time version 1.0 (and bug fix version 1.1 that was released a week later) was available for download. A number of downloads were from private companies that were doing industrial-scale development in Maemo environment.

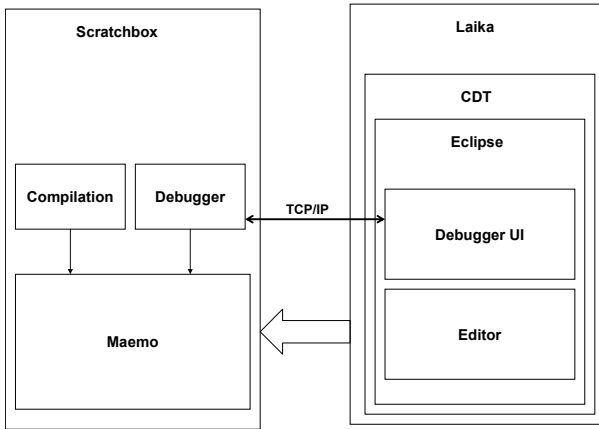


Fig. 2. Laika Release 1 components

During this development period, the main challenge of the project was to be able to create the first running version of the system, because the component systems originating from different communities were not straightforward to integrate. A part of the success is due to the technical skills of people who had already started the development work at Nokia. However, the effort invested in the development by the students should not be underestimated.

3.2 Peak of Fame

Upon completing the first public release of Laika, further interested stakeholders were identified. To begin with, a local company wanted to participate in the development of the system, and they agreed to integrate a user interface editor to the system, a task that the community happily welcomed them to perform. Secondly, project funding was provided to the community both from companies and by the university hosting the project in exchange for using the project as a guinea pig for open source related research. Funding from both sources was gladly accepted by the project, and used for further development of the system. In practice, the three students who had already worked on the system were now all hired to maintain and further develop the system by the university.

The additional funding and support described above kept the project running from September 2005 to January 2007. During this time period, a number of improvements were added to the system, including the above-mentioned UI editor, which was added by the company, as well as support for Python, a dynamic scripting language with which it is easy to compose small applications. The resulting system, illustrated in Fig. 3, required some changes in the tool architecture, as available open source components were based on different designs. In addition, a number of smaller enhancements were made, that in general eased application development.

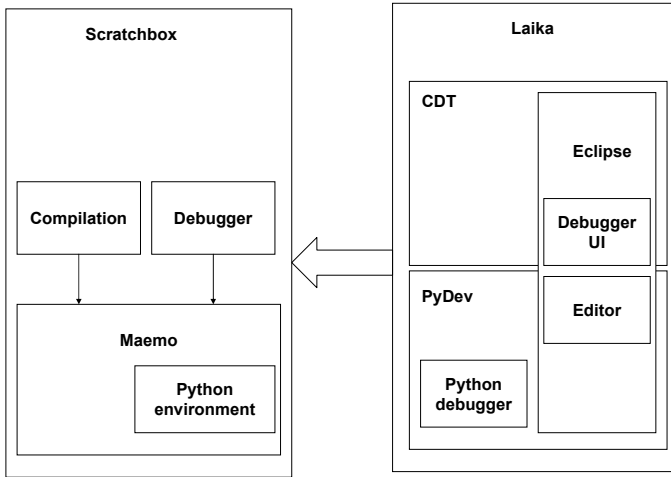


Fig. 3. Laika Release 3 components

At this point of the development, the downside of using a collection of existing components started to manifest. Creating a new configuration where some versions of component subsystems were supported often required additional coding and testing, which seemed to offer little reward for developers.

In general, a lot of external interest was paid to the community at this point, including a number of accepted papers [9][10] as well as a presentation regarding the project at one of the collocated sessions at GNOME User and Developer European Conference (GUADEC) 2006 [16].

In this phase, the main challenge of the community was to cope with ever-increasing number of updates in subsystems that comprise Laika. As already described in [10], new releases of other systems forced the community to perform regression testing of Laika repeatedly. Another issue that can be considered important is user support, which was found a difficult task to perform well. The reason is that most of the issues were related to component subsystems, not Laika as such, and we felt that to some extent this task was beyond our community’s scope. In fact, in some occasions, the community ended up supporting people in installing and using Eclipse for instance in order to allow them to use Laika.

3.3 Decline

By January 2007, project Laika and the related community started running out of steam. To begin with, one of the three students that were originally participating in the development had left the project, and the other two were just about to complete their thesis about the experiences with Laika [11]. Generous funding from different sources was terminating. Moreover, also the project manager and the responsible professor had assumed other tasks and responsibilities, and it was not an option for

them to take a leading role in code-level activities in Laika. A lead developer, on the other hand, was considered a necessity for the project to remain vital.

Despite the fact that the future of the community started looking weary, a number of positive things was happening. The community still gained a lot of interest, and there have been active negotiations on receiving support from involved companies, either in the form of funding or developers. Recently, Laika's code base was moved from university's server to a new location in Maemo Garage [12], a high-visibility web site hosting Maemo related material.

The essential question was how to continue with the development and to establish a real community instead of simply maintaining the project as long as continuous funding exists, and terminate it when the funding ends. At present, there still is interest in Laika, and attracting companies or institutions to participate in development is a reasonable goal. It may even be possible to integrate our system with another similar system, like Esbox [5] for instance, which builds on the same main components as Laika, which in turn would strengthen development. Even more importantly, we need to find a suitable long-term lead for the community – preferably one of the people who have already participated in the development. Motivating the lead, however, might require external attention or in the best case secured funding over some period of time.

3.4 Epilogue

Around the same time the first complete version of this paper was composed, we decided that due to resource restrictions joining forces with Esbox would be best for the Maemo developers. As a result, a new project was established in Maemo Garage under the name Esbox, which in the end will hopefully put together the best parts of Laika and Esbox. At present, however, core developers of Laika are yet to participate in this activity, and no major role has been assumed by us.

4 Lessons Learned

Obviously, there are numerous issues that could have been implemented differently in the course of our project. However, we trust that the reasons why we were unable to attract other people onboard – the real reason for not becoming a lively community – are few. Matters related to this creation of an active developer community are addressed in the following.

Focused mission which remained undocumented. One of the particularities of open source development is the mission of the community. In our case the mission was to “implement an industry-scale integrated toolset for composing Maemo applications”, which also sets the scope for the community. For instance, it overrules extending the community to systems other than Maemo. In addition, we decided to base our work to existing, readily available open source subsystems. With this as the starting point, we were able to establish the first running system relatively rapidly,

which in turn gave us instant credibility. Not documenting the mission can be considered a mistake, however. Without manifesting the mission, it has been difficult for other developers to grasp what Laika really is about.

Too much a debugging, integration, and testing project and too little a real development project. Another mistake regarding the mission is that we defined it in terms of what the community would provide, not what the community would implement in the technical sense. This led to two-fold development focus. One part was about developing support in terms of code that was composed by us, whereas another part was selecting and integrating suitable components that readily existed. To satisfy the latter, it was necessary to perform a lot of straightforward debugging and bug fixing when new versions were released. After a while, constant debugging, integration, and testing started to overrule the project. In an ideal world, we could have geared our own implementation into a direction where maintenance and building new releases would have been eased. This never happened, though.

Developers interested in SDK development, not that much in Maemo programming. Almost from the beginning of the project it became obvious that all developers were focused to developing SDK, and that developing Maemo applications was not an important issue. As a result, the development effort started from SDK and features that would be interesting from SDK point of view, not from the development of Maemo applications and easing it except indirectly. The effect of this lack of “eating our own dogfood”, advocated by Spolsky in [20], to the community is hard to estimate. However, as we had another team, working in a neighboring office that was performing development using also our tools, we feel that this shortcoming is not too severe.

Relying on agile principles that are not necessarily ideal for community building. We organized our development following the spirit and principles of agile software development and extreme programming, like pair programming, collective code ownership, and a common development room [2][13]. In addition, the mindset we adopted was to go for a development mode, where code would be the main artifact, and other items would only be written when absolutely necessary. While the approach has worked well in our own development, there were some problematic issues. In particular, as we were busy developing the system ourselves, we were not too open for other potential participants. This resulted in entry barriers for new contributors, which in turn made our community closed. As the main activity of the community was focused in a single office, where all the developers and the project manager were co-located, this office quickly became the mental home of the community. Therefore, in order for others to participate, barriers existed already in the physical environment. A better alternative, at least from the viewpoint of community creation, would have been to rely on practices and processes that have already been documented explicitly in the scope of open source software [17].

Steep allocation of responsibilities but minimal staffing. The allocation of responsibilities in the community can also be considered. Led by a professor and a project manager, a number of students were performing the actual tasks. This unfortunate setting has resulted in a situation where the students had the real technical

control, while the professor and the project manager were mainly focused to obtaining funding, users, and partners. When the students had their theses completed and they moved to different positions around the same time, the project was left without a lead who singlehandedly could manage both technical aspects and external relations.

Subcontracting for money instead of community building for free. While in general considered positive – and probably a major reason for the possibility to establish a community in the first place – the generous funding we obtained also has some downsides. As our plan was to create a self-sustaining community, the development was soon driven by feature requests associated with funding, not with our own plans. This in turn strengthened an attitude that we are actually performing feature-based subcontracting, not establishing a community.

Lack of active marketing, user support, and other related supporting facilities. As pointed out in [8], open source systems require marketing, similarly to any other software whose use is being advocated. Corresponding arguments can be made for other supporting facilities, including user support for instance. This aspect of the community building remained overlooked in Laika. As a consequence, only people that we were able to relate to had the potential to become participants in the community. Moreover, even using the system built in the community required considerable technical skills.

Single client community. In many cases, open source projects can be taken as common solutions for recurring problems encountered by several organizations. In our case, however, we were only aiming at easing the development of Maemo applications, something that only associates with one other actor. While there were several companies that had interest in developing applications, in practice what we believed was the best for Maemo community was something that we in practice had to obey to a great extent.

No experience on establishing an open source project. An issue that we did not consider important when we started was experience on community work. Looking back now, it would have been essential to involve someone with experience on ramping up an open source project in the community. However, while simple on paper, achieving this in practice might have been difficult.

5 Discussion

Considering Laika now, we feel that we got the development technology right, although one can argue that for instance versioning support could be drastically improved. In addition, we were able to create pull from users of our system. They accepted our system as their development environment and in fact approached us with future development ideas. Moreover, we were able to compose a system that did satisfy – and in many cases exceeded – the requirements we initially set to it. Obviously, we owe a great deal of our success to other open source communities, as without using existing components this would have been impossible. However, we were

still unable to attract more developers to help us with our effort, which in the end has led to a decline.

During the development of Laika, we encountered a number of properties that have larger significance for community-based development. Our main observations are listed in the following:

- Community-based development worked well when developing new features. In contrast it was hard to motivate developers volunteering in the project when tasks like integration and testing was repeatedly requested. Building a toolset that would have eased the latter would have been an essential element for the continuation of the community.
- Distributing responsibilities in a fashion that resembled a small project, where managers were focusing on funding rather than coding, led to a community where no lead developer existed. As a result, there is no single actor that could (or would) singlehandedly continue maintaining the project.
- Agile practices that emphasize the importance of colocated development teams were harmful when trying to attract new developers to join the community. Instead, we should have embraced new developers by offering easy ways to practically participate in the development.
- Using the system ourselves might have helped in solving some of the problems associated with our development effort, especially when considering the priorities of different features. As a result, people using the system might have benefited more.
- Allowing supporters – especially those who provide funding – to overly control a community was fruitful in the beginning, as it gave us confidence on our mission. In the long run, however, this led to subcontracting mode, where funding was associated with individual features instead of supporting the community in general.
- It is easy to overextend the development effort, as the number of readily available open source components that can be integrated in the system is so large. However, in order to keep the system manageable in the long run, it is essential to focus on a relatively small number of key features at least in the initial phases when the community is small.

Still, to our knowledge there was no single community at the time aiming at supporting industrial IDE in Maemo environment that would be considerably stronger than our attempt. Moreover, we have received frequent requests for new features and updates, but have been unable to comply with the requests due to resource constraints in community personnel. This exhaustion of resources, more than anything else, can be taken as an indication that we have failed in forging a lively community.

Acknowledgments

The authors are grateful to the whole Laika community as well as all the partners who have participated in our work. In addition, we wish to thank also the users of Laika for their choice of a development system.

References

- [1] Aaltonen, Timo, Järvensivu, J., and Mikkonen, T. OSS architecture and implications. 67-77, *Empirical Insights on Open Source Software Business* (Eds. Nina Helander and Maria Mäntymäki), *eBRC Research Reports 34*, Tampere, Finland, 2006.
- [2] Beck, K. *Extreme Programming Explained – Embrace Change*. Addison-Wesley, 1999.
- [3] CDT homepage. Available on the Internet at <http://www.eclipse.org/cdt/>.
- [4] Eclipse homepage. Available on the Internet at <http://www.eclipse.org>.
- [5] ESbox homepage. Available on the Internet at http://wiki.embeddedacademy.org/index.php/ESbox_Plug-in.
- [6] Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K. *Perspectives on Free and Open Source Software Development*. MIT Press, Cambridge, MA, USA, 2005.
- [7] GTK+ homepage. Available on the Internet at <http://www.gtk.org>.
- [8] Henttonen, K. *Stylebase for Eclipse. An Open Source Tool to Support the Modeling of Quality-Driven Software Architecture*. VTT Research Notes 2387, Espoo, Finland, 2007.
- [9] Järvensivu, J., Helander, N. and Mikkonen, T. Dependencies, Networks, and Priorities in an Open Source Project. 116-125, *Handbook of Research on Open Source Software: Technological, Economic and Social Perspectives* (Eds. Kirk St.Amant and Brian Still), IGI Global, 2007.
- [10] Järvensivu, J., Kosola, M., Kuusipalo, M., Reijula, P. and Mikkonen, T. Developing an Open Source Integrated Development Environment for a Mobile Device. *International Conference on Software Engineering Advances*, Tahiti, French Polynesia, Oct. 29.-Nov.3., 2006.
- [11] Kuusipalo, M. and Reijula, P. *Implementing a Visual Development Environment for Maemo Compatible Devices*. MSc. thesis, Tampere University of Technology, 2007. In Finnish.
- [12] Laika homepage. Available on the Internet at <http://garage.maemo.org/projects/laika>.
- [13] Larman, G. *Agile and Iterative Development*. Addison-Wesley, 2003.
- [14] MacCormack, A., Rusnak, J., Baldwin, C. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. 1015-1030, *Management Science*. Vol. 52, No. 7, July 2006.
- [15] Maemo homepage. Available on the Internet at <http://www.maemo.org>.
- [16] Reijula, P. Integrating Maemo Development Environment with Eclipse. *GNOME User and Developer European Conference (Guadec) 2006 warmup weekend*, Barcelona, Spain, Jun 24, 2006.
- [17] Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K. Understanding free/open source software development processes. *Software Process Improvement and Practice*, 92-105, No. 11, 2006.
- [18] Scratchbox homepage. Available on the Internet at <http://www.scratchbox.org/>.

- [19] Sillanpää, M. Extending Eclipse and CDT for embedded systems development. *EclipseCon 2006*, Santa Clara Convention Center, Santa Clara, CA, USA. March 20-23, 2006.
- [20] Spolsky, J. *Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity*. Apress, 2004.

Mapping Linux Security Targets to Existing Test Suites

C.A. Ardagna¹, E. Damiani¹, N. El Ioini², F. Frati¹, P. Giovannini² and R. Tchokpon³

¹ Department of Information Technology - University of Milan
via Bramante, 65 – 26013 Crema (CR) - Italy
{ardagna,damiani,frati}@dti.unimi.it

² Free University of Bozen-Bolzano

{Pietro.Giovannini,Nabil.ElIoini}@stud-inf.unibz.it

³ Institut de Mathématiques et de Sciences Physiques – Benin
ricotchfr@yahoo.fr

Abstract. The Common Criteria standard provides an infrastructure for evaluating security functions of IT products and for certifying that security policies claimed by product suppliers are correctly enforced by the security functions themselves. Certifying Open Source software (OSS) can pave the way to OSS adoption in a number of security-conscious application environments. Recent experiences in certifying Linux distributions has pointed out the problem of finding a mapping between descriptions of OSS security functions and existing test suites developed independently, such as the Linux Test Project. In this paper, we describe a mechanism, based on matching techniques, which semi-automatically associates security functions to existing test suite such as the ones developed by Open Source communities.

1 Introduction

Security software evaluation and certification have become an important technique for selecting IT products to be deployed in security-conscious environments. The main goal of software security evaluation is to certify that software products provide and correctly implement some required security functionalities. This means that a good evaluation criterion should give an assessment of system security revealing all the security problems and weaknesses of IT products, which could lead to any unexpected behaviour.

The first attempt (carried out in 1985) to create a standard for security certification was the Trusted Security Evaluation Criteria (TCSEC) by U.S Department of Defense, also known as *Orange Book*. The Orange book has been defined to assist the design and development of security requirements and to fill the communication gap between vendors, evaluators, and customers [16]. However, the Orange Book soon proved too rigid to adapt to the new changes.

Please use the following format when citing this chapter:

Ardagna, C.A., Damiani, E., El Ioini, N., Frati, F., Giovannini, P. and Tchokpon, R., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 29–45.

From 1990, a specific European standard known as *Information Technology Security Evaluation Criteria* (ITSEC) has been adopted in Europe [20]. A few years later, several countries like Germany, United Kingdom, France and Canada have introduced their own national standards for security evaluation. The diversity of certifications standards, however, made the process of certifying an IT product at an international level somewhat difficult and expensive.

More recently, the *Common Criteria* (CC) certification standard has been defined to fulfil the needs of an international standard for affordable software security certification. Common Criteria is now an ISO standard (ISO 15408); it provides an unified process and a flexible framework to specify, design, and evaluate the security properties of IT products [9]. The CC standard is the result of a joint effort of many international organizations that have worked for two decades to specify a common structure for evaluating security properties of IT products [9]. A major goal of the CC evaluation is to certify that the security policies claimed by the developer are correctly enforced by the security functions of the product under evaluation.

Today, many development communities and other organizations working on Open Source software (OSS) recognize that a standard way to obtain the security certification of an OSS product can pave the way to its adoption in a number of security-conscious application environments.

In this paper, we focus on CC certification testing, which is aimed at ensuring that security functions under evaluation are correctly implemented and work according to the specification. In this context, one of the main problems to be faced is to find an easy map between test suites and security functions. This is especially true in OSS scenario, where security functions developers are often different from test suites developers. Recent experiences in certifying Linux distributions [18,19] has pointed to the problem of finding a mapping between descriptions of OSS security functions and existing test suites developed independently, such as the Linux Test Project. Semi-automatic mapping allows, using existing test suites, to support certification and may be the only possibility when tests are developed and made available independently with respect to source code. In the fullness of time, our approach will support a mechanism based on semantics-aware test descriptions for automatically associating security functions to existing test suites.

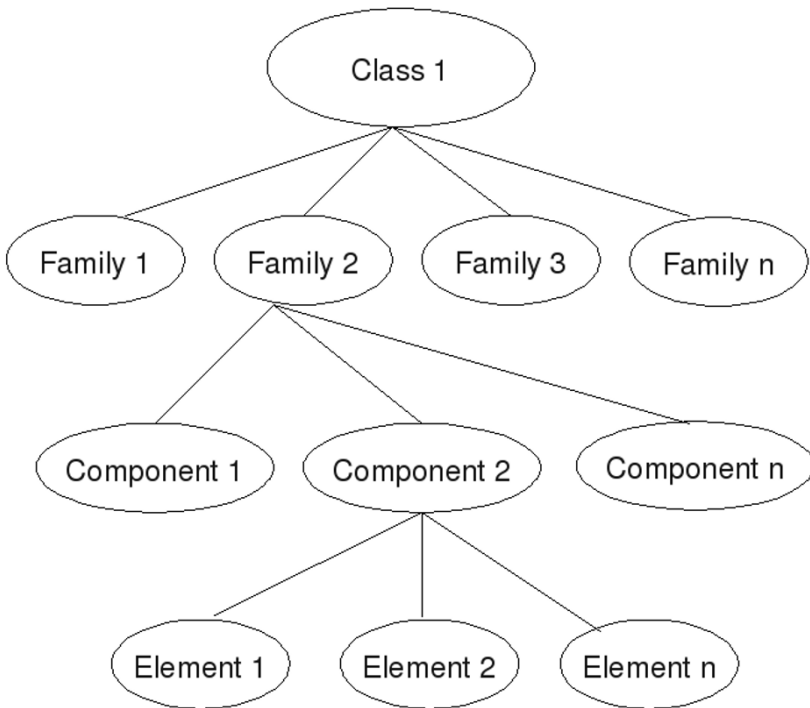


Fig. 2.1 The hierarchical structure of the SFRs.

2 Structure of Common Criteria Certification

The CC certification is based on different components that fall in the following categories [8].

Security Functional and Assurance Requirements.

The CC certification defines two types of security requirements: *Security Functional Requirements* (SFRs) and *Security Assurance Requirement* (SARs). SFRs define the requirements the security functions of the product under evaluation should satisfy [8]. The CC standard includes a predefined extendable catalogue of security functional requirements “that are known and agreed to be of value by the CC part 2 authors” [6].

SFRs are organized in a hierarchical structure, as depicted in Fig. 2.1 [9].

SARs describe practical ways to check the effectiveness of the security functions of the product under evaluation [18]. The SARs catalogue includes many predefined requirements focusing on different phases of the product life cycle such as development, configuration management, testing and so forth.

Protection Profile and Security Target.

A Protection Profile (PP) is a document produced by consumers groups and communities that describes its authors' security needs.

A PP is not meant to be referred to a specific product (i.e., the Target of Evaluation (TOE)), but rather to a product type (i.e., a TOE category). PP defines an implementation-independent set of IT security requirements for a category of TOEs, to be used as a starting point for the evaluation of a particular IT product.

Users can create their PP based on their high-level security needs, without looking at any implementation detail [9]. A Security Target (ST) instead contains the security requirements of a given IT product, to be achieved by a set of specific security functions.

The ST is a basis for agreement between the developers, evaluators and, where appropriate, users on the TOE security properties and on the scope of the evaluation.

A ST can be derived from a given PP by instantiation; in general, the construction of a ST corresponds to a particular PP definition. A ST may then claim conformance to a PP by providing the implementation details concerning the security requirements defined by that PP [12]. Also, ST may augment the requirements derived from the PP.

There might be cases where there is no PP that matches the security properties of a specific product. In this case, the product developer can still create its own ST without claiming conformance to any PP [12].

An important aspect about ST requirements specification is the definition of the *threats* and *security objectives* of the TOE and its environment. In particular, threats identify situations that could compromise the system assets, while *security objectives* contain all the statements about the intents to counter identified threats and/or satisfy identified organization security policies and assumptions. Based on threats and security objectives, the ST defines the *security requirements* that the TOE security functions need to satisfy to achieve the security objectives.

Evaluation Assurance Levels.

The Common Criteria standard defines seven hierarchical Evaluation Assurance Levels (EAL), which balance the desired level of security and the cost of deploying the corresponding degree of assurance [3]. EALs identify different sets of security assurance requirements, which are shown in Table 2.1. In case the predefined requirements do not match the level of assurance required, the EAL might be augmented by adding additional assurance requirements.

Table 2.1 Evaluation Assurance Levels.

EAL	Description
EAL1	Functionally tested (black box testing)
EAL2	Structurally tested
EAL3	Methodologically tested and checked
EAL4	Methodologically designed, tested and reviewed
EAL5	Semiformally designed and tested
EAL6	Semiformally verified design and tested
EAL7	Formally verified design and tested

3 The Linux Security Target

As mentioned earlier, a PP defines a security target template directed to particular application domains, describing the security needs of consumers. This potentially results in different PPs listing different requirements for the same type of products. For instance, two groups of users of the same operating system may have different security needs. The first group could require the operating system to provide secure authentication and authorization mechanisms, while the second group could require the operating system to include a secure data transmission mechanism. These two groups of consumers would construct or cite two different PPs for the same product type, based on their expectations.

In this paper, we focus on the certification of Suse Linux Enterprise Server V8 distribution with service pack 3 (SLES8). The ST adopted for SLES8 claims conformance to the Controlled Access Protection Profile (CAPP), which has been released by the Information Systems Security Organization (ISSO) as part of its program to promote security standards for information systems [17]. According to the ISSO “CAPP-conformant products support access controls that are capable of enforcing access limitations on individual users and data objects CAPP-conformant products also provide an audit capability which records the security-relevant events which occur within the system” [17].

In the following of this section, we discuss the SLES8 ST, describing its main components and giving some examples as reported in the SLES8 ST.

Threats, Objectives, and Security Requirements.

The threats part of a ST identifies the threats that could compromise the system assets. In the case of SLES8, the assets to be protected include the information stored, processed or transmitted by the TOE (SLES8) [11]. The threats are generally divided in two categories: *i*) threats to be countered by the TOE, which exploit weaknesses in the TOE itself, and *ii*) threats to be countered by the TOE environment, which exploit the weaknesses of the TOE environment.

Examples of threats to be countered by the TOE and by its environment are, respectively [11]:

- an authorized user of the TOE may access information resources without having permission from the person who owns, or is responsible for the information resource for the type of access;
- an attacker with legitimate physical access to the hardware of the TOE may cause a hardware malfunction with the effect that a user is losing stored data due to this hardware malfunction.

Security objectives provide statements about the intents to address and counter the identified threats [9]. Based on the above examples, the following security objectives for the TOE and TOE environment, respectively, can be defined [11]:

- the TOE security function must control access to resources based on the identity of users, it must also allow the authorized users to specify which resources may be accessed by which users;
- those responsible for the TOE must ensure that those parts of the TOE critical to security policy are protected from physical attack.

Once the threats have been identified and the objectives to counter them have been set, the ST specifies how those objectives will be achieved by defining the security requirements. Security requirements section provides all the details concerning each of the SFRs selected to reach the security objectives. An example of a SFR provided by the SLES8 ST is: “the TOE security function shall be able to include or exclude auditable events from the set of audited events based on the following attributes [11]: *i) User identity, ii) System call number, and iii) Directory or file name.*”

The SLES8 ST includes seven security functions to be evaluated with respect to SFRs: *identification and authentication, audit, discretionary access control, objective reuse, security management, security communication, and TOE security function protection* [11].

4 Associating Tests to Target

One of the main responsibilities of CC security evaluation teams is to ensure that the security functions claimed by developers have been implemented and are being correctly enforced. To this aim, security development and evaluation teams execute a set of tests to check each security function of the TOE. However, to be able to carry out this task successfully, security teams need to know in advance which tests will be used for which security functions. In other words, a mapping between security functions and test cases is necessary. When products are developed from scratch following the CC standard, developers can create a mapping between the security functions

and their test cases during the test phase of the product life cycle. For existing products, however, security development and evaluation teams need to find out which existing tests match the desired security functions.

This problem is hard in general, as test documentation can be terse or entirely lacking, and is even harder in case of OSS evaluation. OSS development activities are not tracked, configuration management is skeletal and in many cases developers of the system functionalities do not write test cases. Therefore, finding the right mappings becomes a time-consuming task.

Our approach is aimed at providing a general automatic solution, based on matching techniques, for associating ST's security functions to tests. In the following, we show how our solution is used to map SLES8 security functions to Linux Test Project existing test suites. We relied on an extended set of test suites that has been used for SLES8 EAL3 certification [11].

4.1 The Linux Test Project

A fundamental aspect of CC certification is represented by extensive testing of the security functions of the TOE, using tests which cover all the range of security functions under evaluation.

The Linux Test Project (LTP) [15] defines a set of tests for Linux with a huge amount of security functions-related tests, which simplify the certification of Linux kernels.

LTP, which started in 2000 with one hundred test programs developed by Silicon Graphics Inc. (SGI), has been developed to improve the Linux kernel by providing automated testing of kernel functionalities [10]. LTP represented a “revolution” in Linux testing and assurance, since no formal testing environment was previously available to developers. The major advantage provided by the development of LTP was not the definition of batteries of tests, but the provisioning of a framework for systematic integration of testing activities. Furthermore, LTP includes an environment for defining new tests, integrating existing benchmarks and analyzing test results.

Nowadays, the latest version of LTP contains more than three thousands tests. In addition to the test suites, LTP also provides test results, a test tools matrix, technical papers and *HowTos* on Linux testing, and a code coverage analysis tool [15].

LTP tests cover a wide range of kernel functions, including system calls, networking and file system functionalities, and their results are restricted to *PASS* or *FAIL*. LTP provides a testing environment simple to install and use, which includes three scripts for executing three different suites of automatic tests [14]: *i) runalltests.sh*, *ii) network.sh*, and *iii) diskio.sh*. To give a clear understanding of a LTP test suite evaluation, in Table 4.1 we provide the summary report of the evaluation of Linux kernel 2.6 on a 64 bit Intel Itanium architecture (formerly called IA-64)¹.

¹ The complete evaluation is available at http://surfnet.dl.sourceforge.net/sourceforge/ltp/ltp-full-20071130_ia64_output.html.

Table 4.1 LTP summary report of Linux kernel evaluation.

Test Summary	Pan reported some tests FAIL
LTP Version	LTP-20071130
Start Time	Tue Dec 4 02:11:29 PST 2007
End Time	Tue Dec 4 03:11:52 PST 2007
Log Result	/root/subrata/ltp/ltp-full-20071130/results
Output/Failed Result	/root/subrata/ltp/ltp-full-20071130/output
Total Tests	849
Total Failures	0
Kernel Version	2.6.16.21-0.8-default
Machine Architecture	ia64
Hostname	elm3b159

In our context, LTP is used to evaluate the security related functionalities of Linux kernels with respect to CC certification.

As an example, starting from LTP, test suites for the CAPP/EAL3+ certification of SuSE Linux Enterprise Server 8 has been implemented [11]; test cases have been written by IBM, SuSE, and atsec, or taken directly from the LTP.

A detailed test plan has been produced to test the functions of SLES8 on each evaluated platform, and such plan includes an analysis of the test coverage, the functional interfaces tested, and the testing against the high level design.

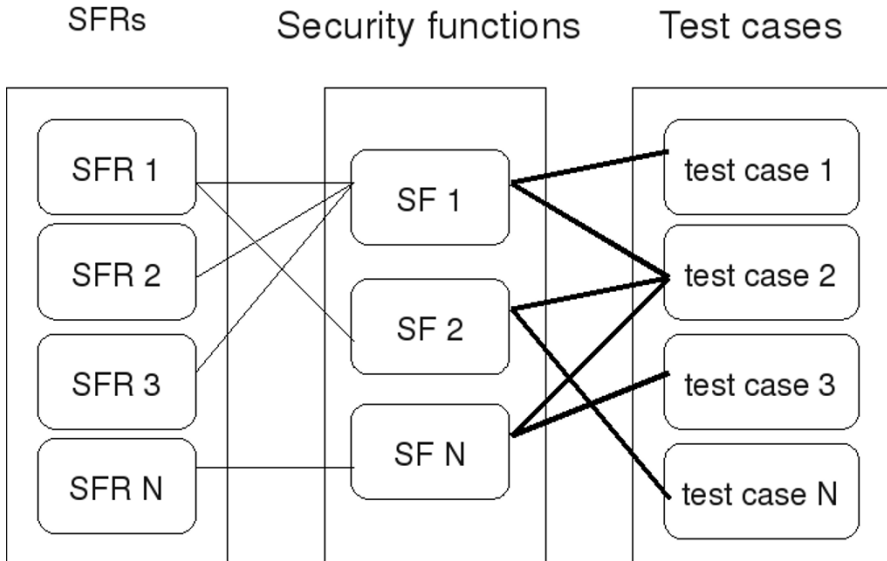


Fig. 4.1 Mapping between SFRs, security functions and testcases.

LTP has been used to test more than fifty kernel versions, where more than five hundreds vulnerabilities have been found. Nevertheless, LTP lacks of adaptability from a security certification point of view, that is, it lacks of an effective technique for associating batteries of tests to specific security functions to be evaluated. LTP, in fact, makes available only three pre-configured scripts for a general purpose testing, making difficult for developers to intuitively identify the batteries of tests that should be adopted for testing specific functions. In the remainder of this section, we discuss a possible solution to this problem.

4.2 Automatic Test Selection for Security Functions

Security functions testing, which is aimed at confirming that the functions operate according to their specification and satisfy the SFRs, is an important part of Common Criteria certification.

However, a major problem to be faced in security functions testing is that no framework for automatic linking between test suites and security functions is available. Developers and evaluators are then forced to associate tests to security functions basing on their experience only. To the best of our knowledge, in the past, this task has been done manually by creating two types of mappings. The first mapping connects each SFR with the security function or set of security functions that implement it. For instance, considering SLES8 security target, the security function *User Identity Changing (IA.4)* contributes to satisfy the SFR *User-Subject Binding (FIA_USB.1)* [11]. The second mapping is done between the security functions and available testcases. Figure 4.1 depicts the two mappings. Note that, whereas the first mapping is

part of the TOE security specification section of the ST, the second one is not part of the ST [5].

Our solution focuses on the second mapping and consists of a generic framework for the automatic selection of test suites for the evaluation of security functions. Our framework, depicted in Figure 4.2, is composed of three main components:

- *Certification Test Suites*, which represents a hierarchical structure containing the set of test suites to be used in the certification process. Each node of the structure is assumed to be labelled using a self-explaining name and enriched of metadata describing its semantics. Each leaf-node represents a single test;
- *Security Functions*, which include the functions description of the TOE to be evaluated;
- *Matching Engine*, which is the component responsible for associating tests to security functions.

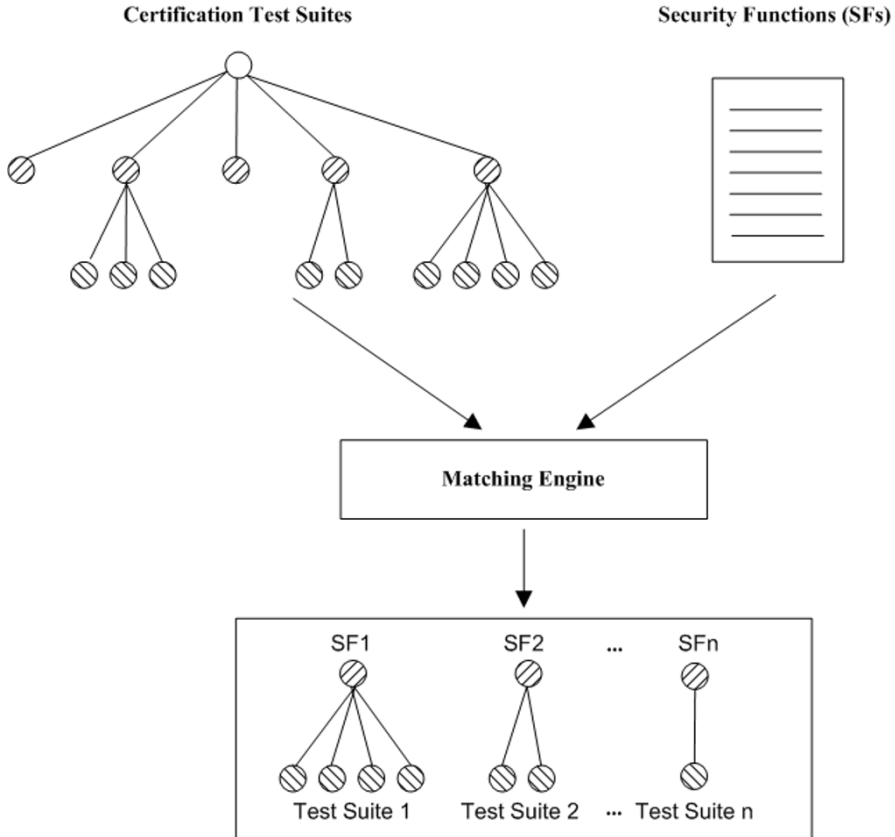


Fig. 4.2 Matching engine.

The first step towards the development of a matching engine for automatic test selection is the introduction of a common vocabulary. A vocabulary can be defined as a set of keywords $\{k_1, \dots, k_n\}$ to be used in the definition of test suites metadata and name, and for the description of the security functions to be evaluated.

The vocabulary V allows to define both certification test suites and security functions descriptions based on a common grammar, which simplify the matching engine process. Our matching engine can be defined as a function f that takes in input the vocabulary V , the security functions SF and the test suites TS , and returns the association between the security functions SF and the test suites TS . The mapping process is composed of three phases: *keyword extraction*, *semantic expansion*, and *search/match*.

In the first phase, given a security function $sf \in SF$, the matching engine searches inside sf all keywords $\{k_1, \dots, k_n\} \in V$ extracting a set $K \subseteq V$ of all the keywords used in sf description. Then, each keyword in $k_i \in K$ is expanded by means of a suitable thesaurus, and the expanded set of keywords is searched in the test suites paths and in each node metadata. All nodes that contain at least one matching, i.e., where at least one keyword is found, are selected and used by the matching engine as the roots of

the subtrees to be considered. Finally, the matching engine returns all the selected subtrees, which are used to test security function *sf*.

Although general enough to be adopted in a generic scenario, our solution is used in the context of the SLES8 Linux distribution. In particular, our framework relies on an extended version of Linux Test Project, which has been used for SLES8 EAL3 certification [11], and on the security functions specification used for the definition of the security target of SLES8.

Since no formal metadata describing the test suites for SLES8 EAL3 certification are available and no agreement between security functions and test suites vocabulary is in place for this application, we assume that the vocabulary used by our matching engine is composed by the set of node names used in the hierarchical structure of the certification test suites, together with some predefined keywords. For instance, the certification test suites used in SLES8 EAL3 certification contains the node path *network/nsfv4/acl* that includes a large set of tests among which: *create_users*, *test_acl*, and many others. All the node names in the path and the test names are used as keywords for searching inside the security functions description and finding the association security functions/test suites.

In the following section, we present a worked-out example based on SLES8 test suites and security target. In particular, after selecting three security functions, we used our engine to select the set of tests to be used for the certification process.

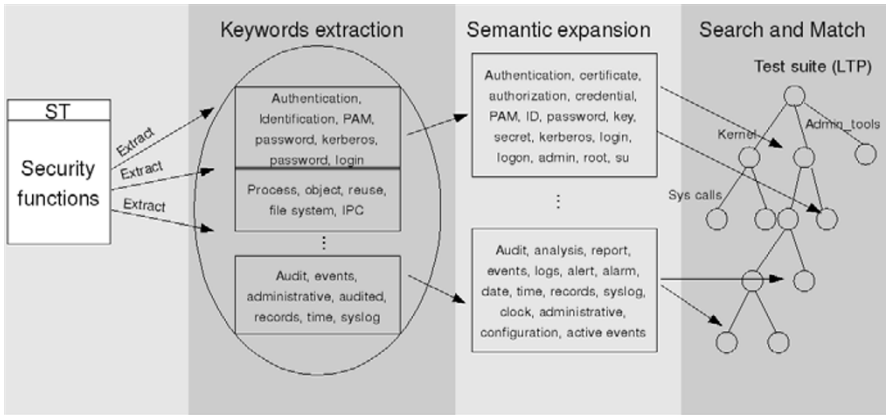


Fig. 4.3 The automated mapping approach between security functions and testcases.

4.3 Automatic Test Selection Example Based on SLES8 Security Functions

The SLES8-based example is summarized in Fig. 4.3 and the results are provided in Table 4.2. During the keyword extraction phase, all the security-related vocabulary keywords are extracted by security functions description (see underlined words in second column of Table 4.2). In the second phase, keywords are semantically expanded using an *ad-hoc* security related thesaurus, generating the column tree of Ta-

ble 4.2. Finally, in the search/match phase, the expanded set of keywords is searched in the LTP testcases directory hierarchy². If a node matches at least one keyword, it will be automatically mapped to the security function to which that keyword belongs (see column four in Table 4.2).

In our approach, the search/match phase relies on the Linux command *grep*, which is applied to the extended LTP tree structure.

The *grep* command, developed within the GNU project, searches one or more input strings and returns any line containing a match to a specified pattern. Based on *grep* command, scripts are generated automatically starting from the keywords retrieved in the semantic expansion phase. These scripts search the keywords retrieved during the keyword extraction and semantic expansion phases, inside the directory structure of extended LTP tree and inside the comments in the testcases. Figure 4.4 presents a sample script, which is executed to retrieve the set of tests for the SC.1 security function described in Table 4.2³.

```
grep -l -i -e 'tunneling' -e 'port 22'
      -e 'secure channel' -e 'secure socket layer'
      -e 'ssl' -e 'ssh' -e 'secure shell'
      -r linux_security_test_suite_EAL3
```

Fig. 4.4 Example of search/match script applied to the SC.1 security function. The *-l* and *-i* options allow to print the list of matching files ignoring case distinctions.

Our experimental results prove the suitability of our solution, since our matching engine always returns all the relevant test cases selected manually by developers during SLES8 certification. Of course, our matching is not entirely precise, and some redundant tests are also selected.

Table 4.2 Automatic test selection results. For sake of conciseness, the table only reports a significant subpart of each security function and a partial list of tests. A complete description of the functions can be found in [11].

Security Function	Description	Semantic Expansion Tests
Access Control Lists supported by SLES (DA.3)	SLES provides support for <u>POSIX</u> type <u>ACLs</u> for the <u>ext3</u> <u>file system</u> allowing defining a fine grained <u>access control</u> on a user basis.	permission, allowed, entry, discretionary, DAC, etc.
Kernel Modules (TP.3)	SLES supports <u>dynamically load-able kernel modules</u> that are loaded automatically on demand. <u>Kernel modules</u> are actually a part of the <u>kernel</u> that is not resi-	access04.c, signal01.c, fcntl07B.c, readdir02.c, etc. Kernel mode, load-able module, modules, trusted program, modprobe etc.
		create_module01.c, create_module02.c, delete_module01.c, delete_module02.c, delete_module03.c,

² Search process considers also the comments inside the testcases files.

³ The complete output produced by the execution of the *grep* command is summarized in Appendix A.

Security Function	Description	Semantic Expansion	Tests
	dent but loaded as part of the <u>kernel</u> when needed.		modules.conf01, modules.conf02, etc.
Secure Protocols (SC.1)	The TOE offers two protocols that applications can use to securely communicate with another trusted IT product (...). Those protocols are the <u>Secure Shell Protocol Version 2 (SSH v2)</u> and the <u>Secure Socket Layer Protocol Version 3 (SSL v3)</u> .	network, secure channel, data transmission, port 22, tunnelling, cryptographic protocol, secure communication, etc.	ssh01, ssh02, ssh03, ssh04, etc.

5 Conclusions and Outlook

In this paper, we outlined a framework aimed at supporting OSS security certifications including a matching engine for the automatic association of tests to security functions. Our solution is a first attempt to provide such an automatic infrastructure, and much additional work remains to be done. In particular, the proposed methodology has the main disadvantage of selecting a redundant set of tests. Our engine in fact chooses all the subtrees of the nodes whose names and/or description match with at least one of the keywords contained in the security functions under evaluation. This results in a situation where unnecessary tests are executed. To overcome this problem, we plan to provide two major improvements in our future work: *i)* the definition of a complete and formal vocabulary, and *ii)* the definition of a formal ontology used to reason about metadata describing the elements to be chosen (i.e., the nodes of the certification test suites hierarchy) and the correlations among them. Such an approach will allow us to exploit automatic reasoning techniques. Also, the combination of a complete vocabulary and a detailed ontology will lead to the development of an engine able to select only the suites of tests that effectively match the specified security functions, thus limiting unnecessary tests executions.

Another important issue is the actual testing environment to be used. It is well known that the development and running of intensive tests, especially for networking applications/devices, could affect the overall consistence of the system under evaluation and of the network that hosts the system. Furthermore, network tests need the participation of at least two physical machines, making them more expensive and demanding more implementation efforts. Also the risk of compromising the integrity of the real system becomes considerable. To eliminate the risk and to reduce the efforts introduced by such an extensive testing, a possible solution consists in setting up a *Virtual Test Environment* (VTE) that accurately reproduces the real environment in which the target application should be evaluated.

Several VTEs have been already provided in literature in different contexts and aimed at several goals [1,2,4,7]. Among them, we have provided a multi-purpose virtual environments, which supplies Virtual Laboratories for the on-line degree courses on Information Systems and Network Security of University of Milan [1,2]. This in-

frastructure also allows the use of the virtual systems for general testing and, in particular, for network functions testing. More in details, the system in [1,2] consists of a virtual subnet composed of a three hosts acting as a firewall, an external generic host, and an internal host working under the firewall. In our future work, we plan to adapt such a virtual network to make it suitable for security testing of applications that should be evaluated to achieve security certification.

Acknowledgments This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591, and within the SecureSCM project in the FP7-ICT Programme under contract n.AOR 213531 and by contract/grant sponsor FIRB research fund of MIUR, research project TEKNE (contract/grant n.RBNE05FKZ2).

6 References

1. Anisetti M, Bellandi V, Colombo A, Cremonini M, Damiani E, Frati F, Hounsou JT, Rebecani D (2007) Learning computer networking on open paravirtual laboratories. *IEEE Transactions on Education*, 50(4):302-311
2. Anisetti M, Bellandi V, Damiani E, Frati F, Raimondi U, Rebecani D (2006) The open source virtual lab: a case study. In *Proc. the Workshop on Free and Open Source Learning Environments and Tools - FOSLET 2006, Como, Italy*
3. Caplan K, Sanders JL (1999) Building an international security standard. *IEEE Educational Activities Department*, 22(3):29-34
4. Gambit Communications. Mimic virtual lab CCNA (2007) www.gambitcomm.com/site/products/vlab_ccna.shtml. Accessed December 2007.
5. The International Organization for Standardization and the International Electrotechnical Commission (2007) *Common Criteria for Information Technology Security Evaluation, Evaluation methodology*
6. The International Organization for Standardization and the International Electrotechnical Commission (2007) *Common Criteria for Information Technology Security Evaluation, Part 2: Security functional components*
7. Grigoriadou M, Kanidis E, Gogoulou A (2006) A web-based educational environment for teaching the computer cache memory. *IEEE Transactions on Education*, 49(1):147-156
8. R. Harland R (2007) The canadian common criteria scheme. http://strategis.ic.gc.ca/epic/site/ad-ad.nsf/vwapj/CSE_Harland.ppt
9. Herrmann DS (2002) *Using the Common Criteria for IT security evaluation*. Auerbach publications, London
10. Hinds N (2004) Kernel kornet: The Linux test project. *Linux Journal*
11. IBM, SuSE, atsec (2007) SuSE Linux Enterprise Server 8 w/SP3 CAPP/EAL 3+ Certification Test Suite. <http://ltp.sourceforge.net/EAL3.html>. Accessed December 2007
12. ISO/IEC. *Guide for the production of Protection Profiles and Security Targets*, 2004.
13. Katzke S (2007) The common criteria (cc) paradigm. ieeiea.org/iasw/Katzke-CC.ppt. Accessed December 2007
14. Larson P (2002) Testing linux with the linux test project. In *Proc. of the Ottawa Linux Symposium, Ottawa, Ontario, Canada*
15. Linux Test Project (2007) <http://ltp.sourceforge.net>. Accessed December 2007
16. Lipner S (1999) Twenty years of evaluation criteria and commercial technology. In *Proc. of the 1999 IEEE Symposium on Security and Privacy*
17. Information Systems Security Organization (1999). *Controlled Access Protection Profile version 1.d*

18. Shankar KS, Kirch O, Ratliff E (2004) Achieving capp/eal3+ security certification for Linux. In Proc. of the Linux Symposium, volume 2:18-30
19. Shankar KS, Kurth H (2004) Certifying open source - The linux experience. *IEEE Security & Privacy*, 2(6):28-33
20. Shi W, Sun Y (2001) An investigation of cc's contribution to confidence in security. In Proc. of the 2001 International Conference on Computer Networks and Mobile Computing, 333-338

Appendix A

An example of a grep-based search/match phase

In Fig. A.1 we provide the complete output produced by our matching engine when mapping between security function SC.1 and testcases is searched. Based on the following grep-based script, where Fig. A.2 shows the set of testcases to be used in the evaluation of security function SC.1.

```

grep -l -i -e 'tunneling' -e 'port 22' -e 'secure channel'
-e 'secure socket layer' -e 'ssl' -e 'ssh'
-e 'secure shell' -r linux_security_test_suite_EAL3

linux_security_test_suite_EAL3/ltplib_EAL2/testcases/network/tcp_cmds/ssh/ssh04
linux_security_test_suite_EAL3/ltplib_EAL2/testcases/network/tcp_cmds/ssh/ssh01_s1
linux_security_test_suite_EAL3/ltplib_EAL2/testcases/network/tcp_cmds/ssh/ssh03
linux_security_test_suite_EAL3/ltplib_EAL2/testcases/network/tcp_cmds/ssh/ssh03_s1
linux_security_test_suite_EAL3/ltplib_EAL2/testcases/network/tcp_cmds/ssh/ssh02_s1
linux_security_test_suite_EAL3/ltplib_EAL2/testcases/network/tcp_cmds/ssh/ssh02
linux_security_test_suite_EAL3/ltplib_EAL2/testcases/network/tcp_cmds/ssh/ssh01
linux_security_test_suite_EAL3/ltplib_EAL2/testcases/user_databases/lastlog01
linux_security_test_suite_EAL3/ltplib_EAL2/testcases/user_databases/faillog01
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/send_new_rsa_key.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/restore_date.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/rsa_login.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/rsa_test.c
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/dsatest.c
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/aestest.c
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/delete_accounts.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/shaltest.c
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/setup_date.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/rsa_auth.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/server.c
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/setup_accounts.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/rc4test.c
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/dsa_auth.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/stunnel_dsa.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/dhtest.c
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/password_auth.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/send_new_dsa_key.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/stunnel_rsa.sh
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/client.c
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/destest.c
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/test/randtest.c
linux_security_test_suite_EAL3/ltplib_OpenSSL/testcases/openssl/openssl01
linux_security_test_suite_EAL3/laus_test/audit_tools/ssh01_s1
linux_security_test_suite_EAL3/laus_test/audit_tools/au_login
linux_security_test_suite_EAL3/laus_test/audit_tools/ssh01
linux_security_test_suite_EAL3/laus_test/libpam/tests/test_sshd.c
linux_security_test_suite_EAL3/laus_test/libpam/libpam.c
linux_security_test_suite_EAL3/laus_test/pam_laus/pam_laus.c
linux_security_test_suite_EAL3/laus_test/pam_laus/tests/test_sshd.c

```

Fig. A.1 Grep-based script for keyword matching and set of testcases retrieved for SC.1 security function evaluation.

Overview on Trust in Large FLOSS Communities

Etiel Petrinja, Alberto Sillitti, and Giancarlo Succi
CASE – Center for Applied Software Engineering
Free University of Bolzano/Bozen
Piazza Domenicani, 3
I-39100 Bolzano-Bozen,
{Etiel.Petrinja|Alberto.Sillitti|Giancarlo.Succi}@unibz.it
WWW home page: <http://www.unibz.it>

Abstract. The paper presents a survey of mature Free/Libre Open Source Software communities. The main focus of the survey is the collection of data related to the practices of these communities related to trust elements in their products. The survey is carried out using a structured questionnaire about thoughts and practices followed by Free/Libre Open Source Software communities. The survey focuses on the analysis of the development processes adopted by such communities. The results of the survey confirms basic ideas related to Free/Libre Open Source Software and explains in more detail specific issues related to trust and trustworthiness of the Free/Libre Open Source Software development process.

Keywords: FLOSS Communities, Software Quality, FLOSS Development Process

1 Introduction

The survey presented in this paper analyses a set of Free/Libre Open Source Software (FLOSS) projects that are used by large communities. The survey focuses on issues related to the trust that users have in adopting FLOSS products. We are interested mainly in processes, tools, methods, and approaches adopted to develop FLOSS products. The survey is part of a four year EU funded project named QualiPSo (Quality Platform for Open Source Software) aiming at studying methodologies, technologies, and policies to leverage the Open Source Software development to sound, well recognised, and established industrial operations [8].

In the last decade, FLOSS products are increasingly being used. This increase is supported by several factors including the absence of direct license costs and the availability of the source code that allows users to adjust FLOSS products to their specific needs. An important drawback of FLOSS is the lack of quality assurance metrics that prove its validity. In commercial environments, the problem is addressed

Please use the following format when citing this chapter:

Petrinja, E., Sillitti, A. and Succi, G., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 47–56.

in a different way. The concept of quality is often related to the certification of the production process of the producer (e.g., CMMI). This approach can be extended also to the FLOSS production (in particular, for FLOSS developed supported by companies). The survey presented in this paper is trying to find out which elements contribute to the trustworthiness that people have in FLOSS products. Trust is linked to both characteristics of the final product and the overall development process followed. In particular, the process is an element that may vary considerably among different FLOSS projects. The survey collects information about different approaches and synthesises them to identify benefits and avoid pitfalls of FLOSS development.

2 Background

In the last decade, intensive research has been carried out on FLOSS. Benchmark results have provided a list of characteristics that are common in the FLOSS development process. The most important are the following [1]:

- it is parallel, rather than linear,
- it involves large communities of globally distributed developers,
- it utilizes truly independent peer review,
- it provides prompt feedback to user and developer contributions,
- it includes the participation of highly talented, highly motivated developers,
- it includes increased levels of user involvement, and
- it makes use of extremely rapid release schedules.

Moreover, the whole FLOSS life cycle differs from the classical software development processes. Mockus *at al.* [7] and Jorgensen [4] proposed similar models identifying phases that are present in FLOSS development projects. The Jorgensen's model [4] includes the following list of phases: code, review, pre-commit test, development release, parallel debugging, and production release.

FLOSS development is strongly influenced by software development tools used by both developers and contributors. In contrast to the diffusion of Computer Aided Software Development (CASE) tools used by traditional software development, FLOSS process adopted software tools as issue/bug trackers, mailing lists, forums, collaboration environments, etc. An important collaborative development environment – SourceForge [3] – provides an integrated environment where more than 100.000 FLOSS projects are being stored and developed (even if only a small portion of them are active projects involving several developers). FLOSS uses also traditional software engineering tools, however not all tools are available as FLOSS products. Therefore, in the near future these tools will eventually appear as FLOSS and will further improve the FLOSS process.

Many researchers have studied the quality and trustworthiness of FLOSS products and their development processes. Since quality is a fundamental ingredient of software and a relevant criterion for adopting FLOSS products, the research was oriented toward the evaluation of the current quality of FLOSS products and how to improve it. Lipner [5] explored the benefits and possible pitfalls of FLOSS development. McGraw [6] stated that “openish” products will not improve security of the software. Schneider [9] stated that source code inspections are just one of possible approaches to improve software’s quality by discovering bugs. Witten [11] explored economics of FLOSS products, metrics used to assess it, and models available.

3 Research Design

The survey includes two parts: one related to FLOSS products and the other related to FLOSS processes. Results presented in this paper deal only with the second part of the questionnaire.

4 Scope

The survey includes seven well-known FLOSS projects: Apache HTTP Server, Eclipse, Emacs, Linux Kernel, Mozilla project, GNOME, and Debian.

5 Methodology

The survey uses the approach proposed by Silverman [10]. It requires the design of a structured and formal research involving two basic and partially correlated concepts:

- The methodology, that is, the specific technique for gathering data (survey, interview, questionnaire, case study, etc.).
- The method, that is, whether performing a quantitative or a qualitative investigation.

Such decisions are based on an evaluation of the goals of the research and the kind of information required.

Our study focuses on gathering opinions about the FLOSS process and products adopted in the surveyed communities. Therefore, we have to conduct a qualitative investigation that involves the analysis of data such as words and sentences instead of numbers [10]. Our research methodology is based on a semi-structured questionnaire filled in mainly by analysing projects’ web sources such as web pages, CVS repositories, mailing lists, and forums. Moreover, some data comes from face-to-face or telephone interviews.

5.1.1 GQM

The overall structure of the research is based on the GQM approach [2], as follows.

- Goal: Evaluate the actual adoption of FLOSS in the software industry.
- Question: The questionnaire is composed of 53 questions with additional sub-questions.
- Metrics: Metrics about the level of adoption and the trust in FLOSS products.

5.1.2 Questionnaire

The questionnaire is organized in 16 sections dealing with different topics related to the FLOSS development processes. The 16 topics are the following:

1. Personal information
2. Company information
3. Role of the organization with respect to FLOSS
4. Issues that can be taken into account when deciding whether to adopt FLOSS
5. Trust
6. Quality assurance
7. General questions
8. Roles and responsibilities
9. Architecture definition
10. Development techniques and practices
11. Tools used
12. Features to implement
13. Documentation and bug management
14. Version control and people management
15. Business model
16. Workflows of the processes identified

The answers gathered through face-to-face interviews are collected following the following steps:

- The respondents are contacted to determine their general interest in the study.
- The questionnaire is sent to the respondents to verify the actual availability.
- Data is collected by personal or telephone interviews.
- The results of the interviews are sent to the interviewees for a final check.

Only upon a positive feedback from the interviewee, the questionnaire is considered accepted and the data is processed. Participants are guaranteed anonymity and the information reported is reviewed so that no individual person or company can be identified. The final questionnaire was designed iteratively; during the survey we added just a small number of new questions and in a few cases we have changed the order of questions to improve the focus on specific topics.

6 Results

In the following subsections we present four topics of the questionnaire: quality related issues, stakeholders related issues, technology related issues and business related issues.

6.1 Quality issues

6.1.1 Trust

Important elements to people surveyed are: openness of the whole development process, openness of the planning process, testing and integration builds, presence of intermediate milestones and visibility of the planed and the actual development process. Moreover, communities trust more common elements such as: the quality of the source code, the correct behaviour of the product, its performance, and often the security of the developed product. We have found out that communities try to fulfil classical trust-related criteria first and additionally they attempt to satisfy important FLOSS-related criteria as well. The same is true when they are testing external FLOSS products to be adopted by the community.

6.1.2 Quality assurance

Quality elements considered important by FLOSS communities vary considerably among different communities. The most important elements listed by communities are the following: the planning process used, the development process followed, the compatibility of the licenses used by specific subprojects, the bug/issue reporting and solving procedure, the availability of appropriate documentation, the simplicity of installation, and a proper integration of different subparts of the final product.

Answers related to testing processes of FLOSS products, either developed by the community or adopted by it, is quite homogeneous among the considered communities. The majority bases the testing process on users and developers that produce FLOSS products. Almost half of the communities have specific test teams that provide a defined quality level of the developed products. On the contrary, one quarter of the communities said explicitly that they do not use any specialized test team. Part of the communities employs beta testers selected from the group of their regular users that provide useful bug/issue reports before the product is largely distributed to the public (Fig. 1).

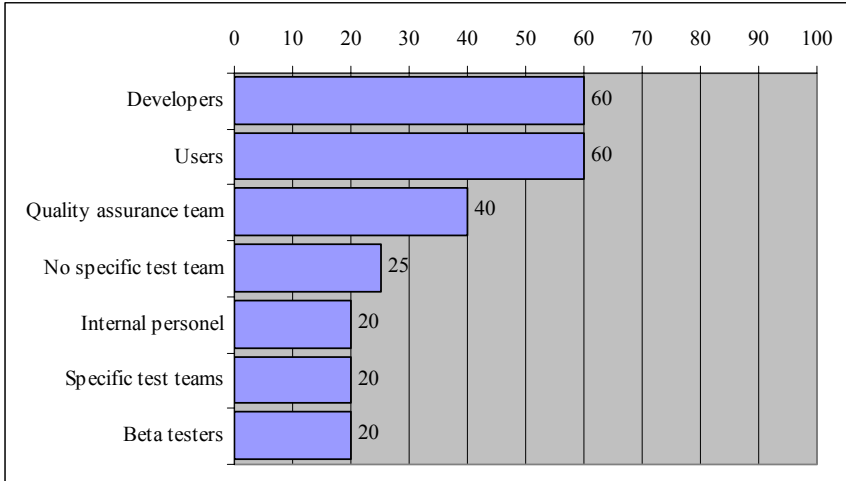


Fig. 1. Percentage of different stakeholders testing FLOSS products.

6.2 Stakeholders related issues

6.2.1 Roles and responsibilities

The number of developers involved in each surveyed FLOSS project varies from 25 core developers to more than 1000 developers. The roles represented in these communities are: simple users, developers, committers, and Project Management Committee (PMC) members. Communities have often an additional management body as can be a technical or a non-technical board that reviews the work done by the community and the progress of the project. Half of the communities surveyed are also supported by a foundation that manages the project and overviews the alignment of the project evolution with basic directives that the project has to satisfy. There is not a common hierarchy structure in different communities. Usually, FLOSS communities do not have many hierarchy levels and they tend to implement a democratic system where everybody has the possibility to express his opinion.

Roles are not always strictly fixed and users can obtain more privileges by providing good quality contributions, stay aligned with local policies and written and not-written rules. In some communities, users can become developers if they provide good quality code and can become committers if their contributions are significant. The role of a user depends on the definition of specific roles in different communities. Often the PMC is responsible for granting new roles and privileges.

6.2.2 Features to implement

New features to be implemented in FLOSS projects are usually proposed in mailing lists or in bug/issue management systems. Features are usually not ranked in the surveyed communities. Exceptions are some bug fixes and features related to impellent architecture modifications. In some communities, new features can be

implemented immediately by developers and contributors, in others, more structured communities, these suggestions can be included in the future implementation plans and roadmaps. The majority of surveyed communities have a time plan for new features to be implemented in the following few months. Usually, plans are available also on web pages to allow everybody to see which features will be added in the near future. In this way, users can participate to the implementation process with source code or documentation contributions.

When changes are proposed, the PMC, the supporting foundation or the responsible person for a specific module decides which features will be implemented. Usually, in the more hierarchically structured communities surveyed there is a specialized development team that implements new features. This team is composed by developers and committers that are responsible to the PMC or to the owners of specific modules. In less hierarchical communities, developers and committers may decide which new features they would like to implement. The possibility to choose which features a developer will implement can be an important productivity advantage that FLOSS processes have in comparison to the proprietary ones.

6.3 Technology issues

6.3.1 The overall architecture

The architecture of the system is defined incrementally or, in few cases, it is planned from the beginning. This depends on the nature and on the size of the project. If projects are strongly centralized they usually have a well planned architecture; on contrary, if projects are a collection of smaller modules, the architecture of the system is defined by just few leading rules. In the majority of the surveyed communities, the architecture planning is often the combination of both approaches.

FLOSS projects are often based on very specific standards. It is the case of the Apache HTTP web server that implements the HTTP open standard. Standards implemented are in the majority of the cases open and sometimes also supported and proposed by FLOSS communities. Another important element present in the surveyed FLOSS communities is the interoperability of the software developed with other (FLOSS and commercial) products. Open standards and interoperability issues are usually interconnected.

6.3.2 Tools used

The operating system used to develop FLOSS projects by all the surveyed communities is either Linux or a Unix-like operating systems. Since the projects surveyed have started many years ago, the products have been adapted to several popular operating systems. More than half of them can run on Microsoft operating systems and many on MacOS.

The most frequently used programming language is not Java as we expected from the current mainstream FLOSS development, but C/C++. The main reason for this is the longevity of the projects surveyed that started in the '90s when Java was

not yet so popular. However, the surveyed communities often use also other languages such as C#, Perl, Python, Lisp, and Java.

The number of different programming tools used by the surveyed communities is very large. One FLOSS community reported that they use 820 different tools and libraries for their development. However, the tools used by the majority of the communities are source code management tool such as CVS (but also GIT, Bit Keeper, LXR, etc.), bug/issue tracking tools such as Bugzilla, and mailing lists .

6.3.3 Development techniques and methodologies

Usually, the adopted development techniques are not well described by the surveyed communities. However, they often explicitly write guiding principles that are used inside the community. Such principles are for instance Quality Culture, Collective Reputation, Freedom, Autonomy, and Evolution. Another element present in the majority of communities is the development process divided into distinct phases that are clearly defined by the community. Transitions from one phase to another are open and there are public reviews. Common phases are: Pre-proposal, Proposal, Incubation, Mature, Top-Level, and Archived.

6.3.4 Documentation

A detailed documentation is a very important part of all the projects developed by the surveyed FLOSS communities. Usually, they start a subproject that is responsible of documentation. It can be produced in different forms, most often as user manuals (separate documents), documentation inserted inside the code (JavaDoc or similar), developer's and maintainer's manuals, and web pages. The creation and the maintenance of the documentation are open to everybody willing to contribute with some effort. Contributors can come also from persons that are not programmers. Documentation is usually protected by a FLOSS consistent license such as Creative Commons.

6.3.5 Bug/Issue management

All the communities surveyed have mailing lists focused on bugs. The majority of communities have also an automatic bug/issue tracking system that helps users and developers to report and manage bugs. More than half of the communities use Bugzilla as the FLOSS bug/issue-tracking solution.

The bug solving procedure depends on the severity of the bug reported, the size of the project, the specific community, and some other issues related to specific bug. The time needed to solve a bug inside communities varies considerably: from one day for 75% of the bugs, up to 140 days for 90% of the bugs in the Apache Server community. Other communities report longer response times for not critical bugs. However, the majority of communities try to provide some answer to critical bugs in one to three days from the notification.

6.4 Business related issues

The business model driving the FLOSS movement does not include revenues obtained from selling licenses for the products. However, there are many additional services, courses, publications, and adaptations that are offered to the market by companies that in many cases collaborate with FLOSS communities. Moreover, there are many indirect benefits that are offered to contributors. Some of them are: the improvement of the reputation of developers, the possibility to get better job positions, working with people that share the same ideas, advancing of FLOSS software in comparison with proprietary software, etc. Some FLOSS communities distribute also grants and prizes that are offered by supporting companies.

A very important aspect that has emerged from various researches done in the last decade on the FLOSS movement, and also from our survey, is the support offered by large software companies to the FLOSS movement. Key developers and leaders in FLOSS communities are often employees of world leading IT companies such as IBM, Hewlett-Packard, Intel, Red Hat, Sun Microsystems, etc. Developers are paid to work on the development of FLOSS products. Companies supporting FLOSS development receives benefits from other sources such as: publicity obtained by contributing to the community, a deep knowledge of FLOSS that can be sold along with proprietary products, attracting good young developers, etc.

7 Conclusions

The results of the survey confirm our expectations on the most important trustworthy elements perceived by FLOSS products developers and users as: the number of downloads, the longevity and the level of activity in the community. The survey revealed additionally more in details which characteristics are essential for FLOSS communities to trust external FLOSS products. The most important aspects of FLOSS development are, as expected, the openness of the whole development process and continuous testing of the product. FLOSS communities try to fulfil first important generic requirements that are guiding also proprietary software development, and additionally they try to accomplish also specific FLOSS related requirements.

Important elements that proof the quality of FLOSS products are: the license used, the quality and completeness of the documentation, and a thorough testing. FLOSS communities often base their testing on specific groups of developers but they rely especially on the large community of users that is an essential part of each FLOSS project. The survey has confirmed also a growing importance that have traditional world leading software companies in further growing of the FLOSS movement. Companies contribute intensively to FLOSS communities usually by paying developers that work for the community. Therefore, many trust related approaches and procedures are often migrated from companies to FLOSS communities.

Standard software development techniques combined with essential FLOSS principles form a higher quality and trustworthy hybrid software development approach. These changes will eventually improve the credibility of FLOSS products and increase their use in companies and public administrations.

8 Acknowledgment

The authors would like to thank the partners of the QualiPSo project that have been involved in the survey. In particular, Prof. Sandro Morasca of the Università dell'Insubria (Italy) who participated to the design of the questionnaire, Prof. Jesus M. Gonzalez-Barahona and Prof. Gregorio Robles of the University Rey Juan Carlos (Spain) who collected the data about the GNOME and Debian communities.

9 References

- [1] The Leading Linux Resource in the World! <http://linux.sys-con.com/>, Last visit December 2007.
- [2] V. R. Basilli: Software modeling and measurement: The Goal/Question/Metric paradigm. Department of Computer Science, University of Maryland, 1992.
- [3] J. Howinson and K. Crowston: The Perils and Pitfalls of Mining SourceForge. University Academic Department.
- [4] N. Jorgensen: Putting it All in the Trunk, Incremental Software Development in the FreeBSD Open Source Project. Information Systems Journal, (2001), 11, 321-336.
- [5] S. B. Lipner: Security and source code access: Issues and realities. University Academic Department, 124-125.
- [6] G. McGraw: Will openish source really improve security? University Academic Department, 128-129.
- [7] A. Mockus, R. Fielding and J. Herbsleb: A Case Study of Open Source Software Development: The Apache Server. University Academic Department, 263-272.
- [8] QualiPSo - Quality Platform for Open Source Software. <http://www.qualipso.org/index.php>, Last visit December 2007.
- [9] F. B. Schneider: Open source in security: Visiting the bizarre. University Academic Department, 126-127.
- [10] D. Silverman: Doing qualitative research. Sage Publications, 2000.
- [11] B. Witten, C. Landwehr and M. Caloyannides: Will open source really improve security? University Academic Department.

PMLite: An Open Source Solution for Process Monitoring

Alberto Colombo, Ernesto Damiani, and Fulvio Frati
Department of Information Technology - University of Milan
via Bramante 65, 26013 Crema (CR) – Italy
{colombo, damiani, frati}@dti.unimi.it

Abstract. Process Monitoring represents a big challenge for organizations that aim to manage software projects adopting different development paradigms. In fact, across-process enterprise-level measurement campaigns can be difficult to enact since process attributes to retrieve are semantically diverse and may be difficult to integrate. In this paper, we present PMLite (Process Monitoring Lite) an open source solution to this problem. PMLite is based on an open metamodel and paves the way to the definition of ad-hoc open monitoring frameworks.

Keywords: open source, process monitoring, PMLite, open metamodel

1. Introduction

Adopting multiple development processes is becoming common in an increasing number of organizations and communities. Different commercial agreements, or different development scenarios, lead to the adoption of a paradigm rather than a different one; as for instance, a development community could use an agile process to develop an open source Enterprise Resource Planning application, where a commercial software house working for a government agency would most probably follow structured Uniform Process (UP) or a waterfall-like process, often formalized in the supply agreement itself.

Such a situation suggests a new vision about software process monitoring: managers need to have a global view of performance, although development activity may be based on different processes that, at a first sight, are incomparable and whose performance data are hard to integrate.

Many research works have attempted a formalization of the notion of software development process and of the associated measurement framework. Piattini *et al.* [7] describes the advantages of using MOF and XMI to represent development processes, giving an overview of MOF and XMI languages and an example of repository for software development process, while Ventura Martins *et al.* [5] presents the *ProjectIT Initiative*, that provides a complete software development workbench and shows an example of development process metamodel. All the approaches above are related to the SPEM (Software Process Engineering Metamodel) specification [4]

Please use the following format when citing this chapter:

Colombo, A., Damiani, E. and Frati, F., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 57–68.

proposed by Object Management Group (OMG), that describes a concrete software development process or a family of related software development processes.

Other existing standard frameworks, such as UML and CWM (Common Warehouse Model) have been used to generate metadata describing complex systems, and can be used for development process representation as well.

Starting from these standards and research works, our group has formalized and tested a metamodel [2] for measuring and assessing generic development process models (see Section 2). In this paper, we highlight the progress of our researches presenting the open source application PMLite (Process Monitoring Lite) [6], available on SourceForge, that fully embraces our methodology. The tool has been developed as a proof-of-concept of our approach, and could be adopted in small medium enterprises that need a lightweight across-process monitoring tool.

The paper is organized as follows. Section 2 provides an overview of the metamodel that defines the structure of the tool, whereas Section 3 describes in details PMLite implementation. Finally, Section 4 shows future extensions of our work and Section 5 draws our conclusions.

2. Defining a Metamodel for Process Monitoring

The first step to define a common environment, for measuring different development process, is to produce a general schema, a *metamodel*, that will describe the underlying structure of processes and the relative measuring framework. We start from the work of Piattini *et al.* in [7], that uses MOF and XMI to represent software processes. In particular, MOF (Meta-Object Facility) [3] is a standard supported by OMG [5] that defines a generic pattern for the construction of systems based on metadata. MOF can be described by its four levels structure: starting from the top there is *i)* the definition of all the concepts and attributes of the language itself, used to build a *ii)* metamodel, that defines the structure and semantic of the metadata related to a generic environment; then, this metamodel is used to create *iii)* models, that depict specific objects and describe the structure for *iv)* the user data.



Fig. 2.1 Our modular meta-model. The yellow, green, and red colors correspond to Process, Measurement, and Trigger modules.

Following the MOF approach, we define the development process and the measurement framework metamodels that are used as basis to model specific development processes and measurement frameworks, and a simple trigger layer to connect the two metamodels. The whole metamodel is presented in Fig. 2.1 where the colors define the individual models.

A complete description of any elements of the metamodel could be found in [2]; for the sake of conciseness, in this paper we limit the description only to the elements that would be directly involved in PMLite development.

Metamodel description.

The three colors in Fig. 2.1 define the three parts of our modular metamodel. It is important to note how the process module is independent from the measurement one, thanks to decoupling via the trigger layer. Such decoupling allows to apply the same measurement framework to projects implementing different development processes, and, consequently, to elaborate across-process assessment.

The yellow blocks identify the development process module. The first node is the element *Organization*, that describe the overall organization and allows enterprise-level measurement campaigns. Each organization manages a set of *Project* classes, each one realized by its own set of *Phases*, that are characteristic of a particular development paradigm. Each phase has its own set of *Activity* nodes, that effectively describes the task put in action during whole process. Furthermore, each phase could be linked to another phase to describe iterative models.

The green module describes the structure of the measurement framework. The framework is based on the *Goal-Question-Metric* (GQM) approach [1], which drives the creation of a measurement system starting first, from the identification of the *goals* of the measures, then of the *questions* that will evaluate them through a set of specific *metrics*. The first element, *Information Need*, is the container node of the module and describes the information need over which the measurement is based and it is used as conceptual link for the two modules. Then, following the GQM approach, the entity *Measurable Concept* defines the areas, i.e. goals, over which the analysis is based. The *Measurable Attributes* node defines the attributes to measure in order to accomplish analysis goals. Further, this element provides the way how these attributes could be gathered; indeed, there is a strict relation between *Work Product* and *Measurable Attribute* classes, since the latter are attributes that could be extracted from the former. The *Measure* class describes the value of the measured attributes and it is strictly related to *KPI* (Key Performance Indicator) and *Metric* nodes, that define an elaboration of the measure instances in order to provide indicators that, respectively, lower the cardinality of the measures and qualitatively evaluate the results.

Finally, the red module isolates the trigger representation, which simply defines the *Trigger* entities, i.e. a plug-in, that physically extracts the attributes values from the work products, storing data in the *Triggered Value* class. As said above, triggers allow modules to be independent one from each other, since they know which attributes to extract and in which work product they have to be physically extracted and in which way.

Instance example: the Scrum agile process.

Our open metamodel has been designed as general as possible, in order to be able to model processes that embrace different paradigms. To demonstrate such property, in Fig. 2.2 we present an instance of the metamodel describing the agile process Scrum [2,8].

We choose an agile development process to show the flexibility of our approach; in fact, agile processes, and in particular Scrum, are intrinsically unpredictable, although a control mechanism is used to guarantee flexibility, responsiveness, and reliability of the results. These characteristics could make difficult the implementation of such a rigorous measurement framework. Thanks to the independence between process and measurement module, our metamodel could seamlessly superimposes a measurement framework to agile-based projects.

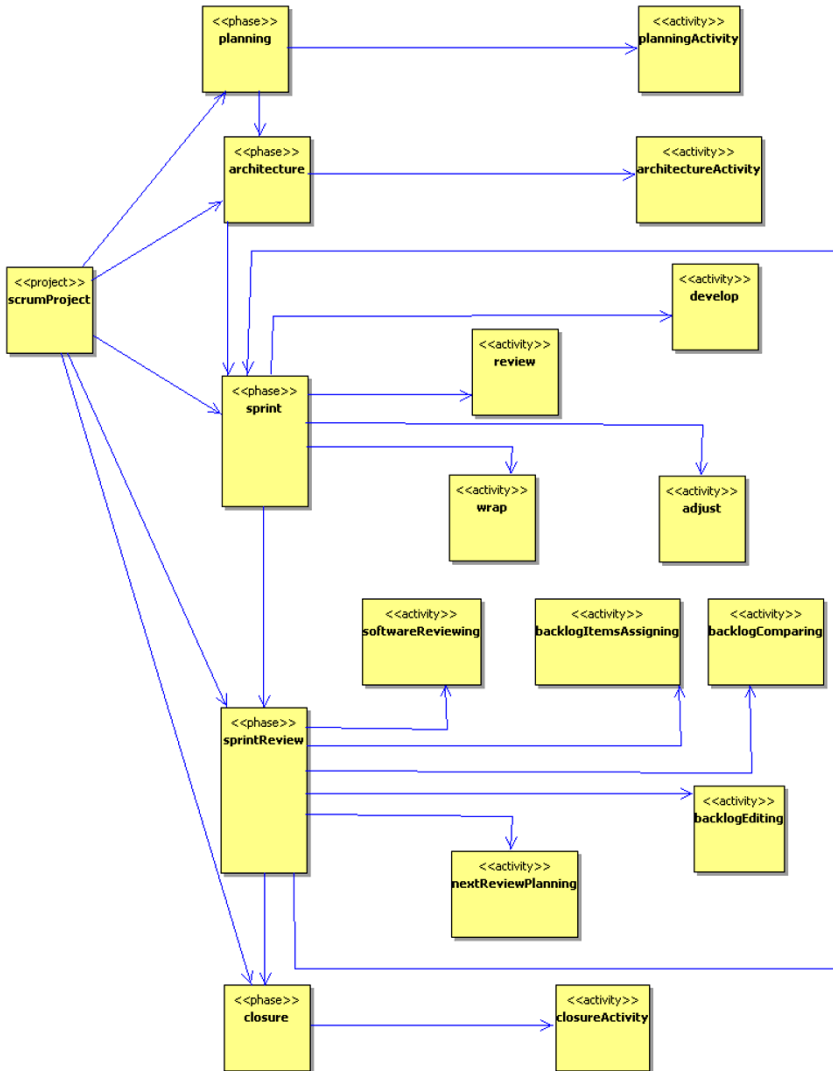


Fig. 2.2 Model of the agile development process Scrum.

3. PMLite: Process Monitoring Lite

The requirements that have driven the development of PMLite are essentially three.

First of all, we wanted to develop a web-based application that fully adopts and verifies our open metamodel representation, allowing managers to model any type of process and organize measurement campaigns to gather all needed attributes.

Secondly, we wanted PMLite to be essentially an easy-to-use tool, with a gentle learning curve, that could be adopted also in small software houses and open source development groups, without any particular installation effort. For this reason, we choose to propose a data collecting technique based on surveys instead of automatic probes, lowering installation problems and development effort. Furthermore, the huge number of applications used during software development and during support activities makes difficult to implement automatic probes that will extract measurable attribute from a suitable set of applications work products.

Finally, PMLite is a first step toward developing a complete process monitoring platform, which could exploit our metamodel approach for generic monitoring generic business processes.

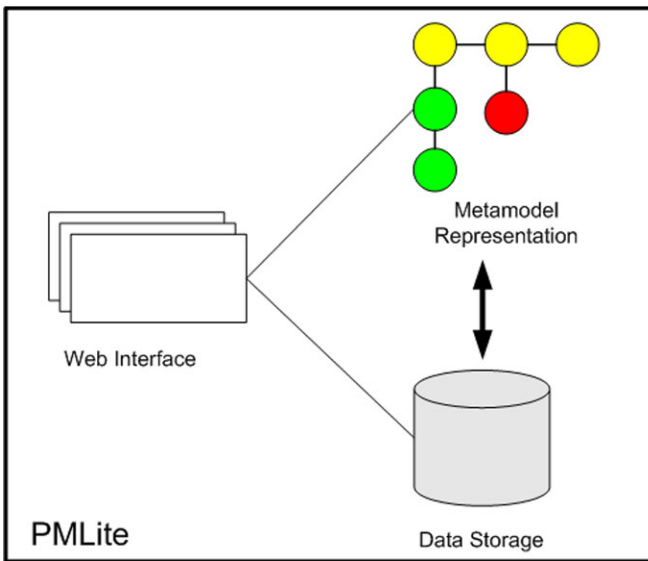


Fig. 2.1 Conceptual structure of PMLite.

3.1. PMLite Description

The conceptual structure of PMLite is depicted in Fig. 2.1. Both web pages and data storage have been designed basing on the metamodel structure and classes, and the supplied activities start from defining, for any project, activities and phases of the relative development process, the measurable attributes to be retrieved, and the questions that compose the surveys.



Fig. 3.2 PMLite homepage.



Would You like to insert a new activity for process " Scrum " ?

Name:

Description:

Name	Description
<input type="radio"/> Develop	Defining changes needed for the implementation of backlog requirements into packets, opening the packets, performing domain analysis, designing, developing, implementing, testing, and documenting the changes.
<input checked="" type="radio"/> Wrap	Closing the packets, creating an executable version of changes and how they implement backlog requirements.
<input type="radio"/> Review	Presenting work and review progress, raising and resolving issues and problems, adding new backlog items.

Fig. 3.3 Interface for the insertion of a new activity.

To better describe the structure of the tool, we concentrate on three key actions: *i)* definition of the process, *ii)* definition of the survey, and *iii)* execution of the survey.

Definition of the process.

The homepage of PMLite (Fig. 3.2) allows the access to specific functions of the application.

The first step is the definition of the specific development processes in terms of phases, activities (see Fig. 3.3), and relationships between phases and related activities. Further, the tool allows to define the transitions between the activities themselves. In this way, the process is well defined and projects can be linked to the specific process.

Then, managers have to define the attributes over which the analysis will be based. The tool makes simple the insertion of measurable attributes (see Fig. 3.4) but the procedure of specification of them is critical, since they will be the basis for the

definition of questions, of surveys, and, consequently, of the whole measurement framework. At the time being, an attribute is only characterized by its name and description.

Definition of the survey.

In its full implementation, the metamodel requires attributes to be retrieved by automatic extractors (i.e. instances of trigger classes). However, for the reasons explained above, PMLite simulates the automatic triggers via specific question sets; users interact with PMLite by answering to the questions associated to the current activity.

PMLite gives specific interfaces to fulfil these actions. In particular, the interface in Fig. 3.5 presents a complete set of questions. Each question is characterized by a text and three possible types of answers (*clear text*, *single choice*, and *multiple choices*). Each question is then gathered in a specific *Question Set*, which, in turn, is associated to a specific process phase or activity. This allows the system presenting to developers the questions sets concerning the specific development action they are performing.



Insert a new measurable attribute

Name:

Description:

Fig. 3.4 Interface for the insertion of a new attribute.

A screenshot of the 'All inserted questions' management interface. It features a list of questions on the left and a configuration panel on the right. The list includes items like 'Requirement name', 'Requirement creation date', and 'Current requirement state' (which is selected). The configuration panel shows 'Measurable Attribute' set to 'requirementState' and 'Alternatives' including 'Modified', 'New', 'Deleted', 'Approved', 'Registered', and 'Completed'. Buttons for 'Modify', 'Delete', and 'View' are at the bottom of the list, and 'Modify', 'Delete', and 'Add' are at the bottom of the configuration panel.

All inserted questions

Questions	Measurable Attribute
<ul style="list-style-type: none"><input type="radio"/> Requirement name:<input type="radio"/> Requirement creation date:<input type="radio"/> Requirement category:<input type="radio"/> Requirement Description:<input type="radio"/> Effort estimation for requirement implementation:<input type="radio"/> Requirement priority:<input type="radio"/> Requirement risk:<input checked="" type="radio"/> Current requirement state:<input type="radio"/> Tool used for requirement management:<input type="radio"/> Component implementing requirement functionalities:<input type="radio"/> Requirement person in charge:<input type="radio"/> Date of last variation:<input type="radio"/> Current requirement version:<input type="radio"/> Indicate the responsible for requirement change:<input type="radio"/> Class name:<input type="radio"/> Number of new lines of code:<input type="radio"/> Number of modified lines of code:<input type="radio"/> Number of reused lines of code:<input type="radio"/> Class complexity:	<p>requirementState <input type="button" value="Change"/></p> <p>Alternatives</p> <ul style="list-style-type: none"><input type="radio"/> Modified<input type="radio"/> New<input type="radio"/> Deleted<input type="radio"/> Approved<input type="radio"/> Registered<input type="radio"/> Completed

Fig. 3.5 Interface for the management of questions.

Execution of the survey.

As said above, each questions set is relative to a specific phase or activity of a process, therefore it is important that the tool will presents to users the questions that are specific of the activity or the phase they are working on. In the interface in Fig. 3.6 first, developers choose the current activity or phase, then, PMLite presents to them the relative series of questions.

The approach followed in developing PMLite, at a first sight, could seem too intrusive, since developers have to manually interact with the survey interface any time they start a new activity or a new process. However, this allows adopting PMLite even in lightweight development environments where no configuration management or event tracking is available.

The screenshot displays the PMLite survey execution interface. At the top left is the PMLite logo with the text 'PM Lite Process Monitoring Lite'. The main heading reads 'Answer to all questions of survey associated to the set' followed by 'Code Question Set'. Below this is a text input field labeled 'Class name:'. At the bottom left is an 'Exit' button, and at the bottom right is a 'Next' button. Progress indicators show 'Set 1 of 1' and '1/9'. The footer contains project information: 'Project: RUP Project', 'Process: RUP', 'Phase: Elaboration' (with a dropdown arrow), and 'Activity: Implementation' (with a dropdown arrow), followed by an 'Ok' button.

Fig. 3.6 Interface for the execution of the surveys.

4. Future Works

PMLite is only the first step in the development of a complete and automatic process monitoring environment, which will be fully transparent and non-intrusive for the developers, but allows us to test and proof our approach.

Our metamodel has been fully exploited for the designing of the structure of a more complete monitoring open source tool, Spago4Q [9].

We plan to exploit PMLite also for the definition and as proof-of-concept of specialized GQM, as for instance for the quality assessment of Open Source products and for the complexity evaluation of business process.

5. Conclusions

In this paper we presented our new open source tool, PMLite, which implements our study [2] of a metamodel to completely formalize an enterprise-level process monitoring framework. PMLite is directed to these organizations that manage projects adopting different development processes and want to have a snapshot of the global status of the current works. The methodology proposed, although could seem intrusive for developers, has the unique strength of being adaptable not only for development process monitoring, but also for generic process representations.

Acknowledgments

This work was supported in part by the European Union within the SecureSCM project in the FP7-ICT Programme under contract n. AO213531, and by contract/grant sponsor FIRB research fund of MIUR, research project TEKNE (contract/grant n.RBNE05FKZ2).

References

- [1] Basili VR (1992) Software modeling and measurement: The goal question metric paradigm. In MD University of Maryland, College Park, editor, Computer Science Technical Report Series, CS-TR-2956 (UMIACSTR-92-96)
- [2] Bellettini C, Colombo A, Damiani E, Frati F (2007) A metamodel for modeling and measuring scrum development process. In Springer Berlin, editor, Agile Processes in Software Engineering and Extreme Programming - Lecture Notes in Computer Science, 4536:74-83
- [3] OMG Group (2008) Mof - metaobject facility. www.omg.org/mof/. Accessed January 2008.
- [4] OMG Group (2008) Spem software process engineering metamodel. www.omg.org/technology/documents/formal/spem.htm. Accessed January 2008
- [5] Ventura Martins P, Rodrigues da Silva A (2005) Pit-p2m: Projectit process and project metamodel. In Proc. of OTM Workshops, Cyprus, 516-525
- [6] PMLite (2007) Process monitoring lite. sourceforge.net/projects/pm-lite/. Accessed January 2008.
- [7] F. Ruiz, A. Vizcaino, F. Garcia, and M. Piattini (2003) Using xmi and mof for representation and interchange of software processes. In Proc. of 14th International Workshop on Database and Expert Systems Applications (DEXA'03), Prague, Czech Republic
- [8] K. Schwaber (1995) Scrum development process. In Proc. of OOPSLA'95 Workshop on Business Object Design and Implementation, Austin, TX, US
- [9] Spago Solutions (2007) Spago4q. www.spago.org/. Accessed January 2008

Requirements Acquisition in Open Source Development: Firefox 2.0

John Noll

Computer Engineering Department
Santa Clara University
500 El Camino Real, Santa Clara, CA 95053 USA
jhnoll@gmail.com

Abstract. Open Source Software Development appears to depart radically from conventional notions of software engineering. In particular, requirements for Open Source projects seem to be asserted rather than elicited.

This paper examines features of the latest major release of the Firefox web browser in attempt to understand how prevalent this phenomenon is. Using archives of mailing lists and issue tracking databases, these features were traced from first mention to release, to determine the process by which requirements are proposed, adopted, and implemented in Firefox. The results confirm the importance of user participation as developers of open source products.

Key words: Innovation, Open Source, Requirements Elicitation, Software Development

1 Introduction

Open source products have garnered significant interest in the popular as well as research literature. Banner projects like Linux, Mozilla, and Apache seem to represent a radical departure from conventional ideas of software engineering, harnessing the labor of numerous volunteers who are distributed throughout the world, contribute according to their desires and abilities, and are motivated apparently only by the joy of creating software, to produce high quality products that dominate their respective markets.

In his essay “The Cathedral and the Bazaar,” Eric Raymond captured this popular conception of the differences between open source and conventional software development, arguing that open source project organization emerges from a marketplace of developers, who gradually take on increasing leadership responsibilities as their contributions to the project are recognised by their peers (the “Bazaar”). This is in contrast to the traditional top-down approach where managers set schedules and assign tasks (like priests in a “Cathedral”) [43].

Research has shown that the perceived differences between open source and conventional software development projects are only partially true, at least for the highly visible projects. For example, the most successful open source software projects em-

Please use the following format when citing this chapter:

Noll, J., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succì; (Boston: Springer), pp. 69–79.

ploy substantial numbers of paid programmers [25]. The Mozilla project has regular face-to-face meetings (augmented by teleconferencing) to review project status [35].

One area where open source software development does appear to differ is in requirements acquisition. Requirements for open source products are often asserted by developers, rather than elicited from users, in contrast to the conventional approach in which user needs are discovered through focus groups, interviews, and other means. This phenomenon has been observed by several researchers studying open source software development [13, 45].

How frequently are requirements asserted in an open source project? This paper presents results of a study of one open source product: the Firefox web browser produced by the Mozilla foundation [30]. The study attempted to determine how many features in the third major release (2.0) of Firefox were actually asserted by developers, rather than derived from user input or other sources.

Of fifteen new features in the Firefox 2 release, nine were determined to be asserted by developers, three were derived from user input, and two features had origins in “extensions” implemented by third parties using Firefox’s extension mechanism.

The rest of this paper is organized as follows: the next section describes the method used to study the product’s features; following that, the study results are presented. The paper concludes with a survey of related work, and conclusions.

2 Method

The Firefox project is interesting for a number of reasons. It has a large user community and is considered to be among the best web browsers available, with superior functionality and security [19]. The Firefox development organization is large and mature, so the processes used for the current release are well established and therefore represent a “typical” way of doing things for a project of this size.

As an open source project, Firefox is unusual in that it incorporates some aspects of more conventional software development organizations: regular (weekly) in-person status meetings are held (although remote participants can contribute via conference calling) [35]; a formal requirements document for Firefox 2 was written [33] and updated periodically [34]; project schedules are created and posted for review [32]. As such, one would expect to see elements of conventional software engineering practices in Firefox development processes, including requirements engineering.

The method employed for this research involved three steps:

1. Identify the set of features comprising the current (2.0) Firefox release.
2. Examine Internet resources related to Firefox, such as archives of discussion groups, web logs, issue databases, and other online forums, to discover when the feature was first proposed, and what role the person proposing the feature played (such as user or developer).
3. Determine the initial implementation of the feature (prototype by a developer, Firefox extension, or enhancement to the core codebase).
4. Categorize the requirement as asserted by a developer, either from his or her personal experience or knowledge of user needs; derived from user contribution, for

example by filing a bug report or “Request for Enhancement” in the Bugzilla issue database; or derived from competition.

The release notes for Firefox 2 (code named “Bon Echo”) list fifteen new features, or enhancements to existing features, for this release:

1. “Visual Refresh” (enhancement).
2. Phishing protection (new).
3. “Enhanced search capabilities” (new).
4. “Improved” tabbed-browsing (enhancement).
5. Resumption of previous browsing session (new).
6. Web feed (RSS) preview and subscription (enhancement).
7. Spell checking (new).
8. “Live Titles” (new).
9. “Improved Add-ons manager” (enhancement).
10. Update to JavaScript version 1.7 (enhancement).
11. Search engine “plugins” in Sherlock or OpenSearch format (new).
12. Updated extension mechanism (enhancement).
13. Support for SVG text (bug fix).
14. New installer for Microsoft Windows©systems, based on the Nullsoft Scriptable Install System (enhancement) [31].

Some of these (phishing protection, session resumption, RSS subscription, spell checking, and “live titles”) represent new features that were not present in previous releases. For these, the first expression of the need for the new feature was used to establish the source of the feature. Other features, such as tabbed-browsing improvements and updated extension mechanism, are modifications to existing features. In these cases, the first expression of need for enhancement (or fix) was used to establish the source of the requirement.

Some “features” listed in the release notes are really collections of enhancements to some category of Firefox behavior or appearance. For example, the “visual refresh” involves changes to icons, colors, and the shapes of buttons and panels. Likewise, the updated extension mechanism involves several changes to the way Firefox incorporates extensions developed by third parties (for example, MultiZilla [42] and Zotero [56]). In such cases, an attempt was made to establish whether the initial need was expressed as a collection, or whether separate needs were later collected under a single category.

3 Results

The results of the analysis are summarized in Table 3. The second column identifies the artifact where the first expression of the requirement was found. In two cases (Enhanced Search and JavaScript 1.7), no mention of the requirement appears before the Firefox Product Requirements Document was published; the source document for these is identified as ‘PRD.’

Table 1: Source and classification of Firefox 2 Requirements

Feature	Requirement Source	Initial Implementation	Classification
Visual Refresh	discussion forum[8, 10]	core	asserted, from knowledge of user needs
Phishing Protection	Bugzilla[48], Internet Explorer[49]	extension (Google Safe Browsing[17])	asserted, from knowledge of user needs and competition
Enhanced Search	PRD[33]	extension (Google search toolbar[18])	derived from extension
Improved Tabbed Browsing	Mozilla Wiki[3, 16], Bugzilla[14]	prototype	asserted and formally validated
Session Resume	extensions (SessionSaver[4], Tabbedbrowser Extensions[21]), Opera	extensions	derived from extension; competition[12]
Web Feed Preview	Bugzilla[2, 28]	extensions (RSS Reader Panel[22], Sage[42])	asserted, from personal experience
Spell Checking	Bugzilla[5, 9]	extensions (SpellBound[51], user contributed Torisugari[53])	
Live Titles	discussion forum[26], developer web log[27]	prototype[26]	asserted, from personal experience
Improved Add-ons Manager	Mozilla wiki[15]	core	asserted, from personal experience
Javascript 1.7	PRD[33]	core	asserted, from personal experience[6]
Search Plugins	Safari[24], Mycroft[36], Sherlock[50], Opera[40]	core	competition
Updated Extension Mechanism	Bugzilla[39]	core	user contributed
SVG Text Support	Bugzilla[41]	core	user contributed (bug)
Windows Installer	Bugzilla[1, 38, 52]	prototype[7]	asserted, from knowledge of user needs

The third column identifies where the first implementation of the behavior appeared in Firefox; ‘core’ refers to the Firefox core codebase; ‘extension(s)’ to a package implemented using Firefox’s extension mechanism that allows third parties to enhance Firefox’s features with substantial functionality; and ‘prototype’ to a developer’s prototype distributed for evaluation by the community.

The last column categorizes how the requirement was proposed; there are five categories:

1. asserted by a developer, from his or her own personal experience;
2. asserted by a developer, based on his or her personal knowledge of what users need;
3. contributed by a user who is not a developer, by posting a bug report or request for enhancement to the Bugzilla issue database;
4. derived from the success of an extension;
5. motivated by appearance of the feature in a competing product.

The majority of requirements (seven total) were asserted by developers, based on either their personal experience or knowledge of user needs.

Three features were influenced, at least in part, by competition from other browsers: Search Plugins are supported by Opera [40] and Safari [24], and are defined by the open-source search plugin efforts Mycroft [36] and Sherlock [50]; Phishing Protection was rumored to be part of an upcoming release of Internet Explorer [49]; and, Session Resume was implemented by Opera [12].

Two features (Phishing Protection and Enhanced Search) were derived directly from successful extension implementations (Google Safe Browsing [17] and Google Search toolbar [18], respectively).

Three features were derived from user contributed bug reports or requests for enhancement: Spell Checking [5, 9], Updated Extension Mechanism [39], and SVG Text Support [41].

The distinction between requirements asserted by developers, and requirements elicited from users in the textbook manner, is not as extreme as it might appear. In the case of Firefox, developers are users as well, so the difference is more one of how requirements are validated than from where they originate: in the Firefox project, requirements asserted by developers are rarely validated by a formal process; in only one case – Improved Tabbed Browsing – was any formal validation process undertaken to confirm that users did indeed need the feature [16].

Rather, requirements are typically posted to discussion forums for informal validation through feedback from other developers, and any dedicated users that participate in the forums. As a consequence, it appears that an open source project like Firefox must have a certain number of developers who are also users, in order to provide accurate requirements: since developers are the source of requirements, enough developers must be in touch with the larger user community to keep the feature set relevant to the bulk of users.

In a few cases, features were initially implemented as extensions, rather than as part of the Firefox core codebase; these extensions proved popular enough, by virtue of the number of people who installed and used them, to be considered for incorporation

into the core. Similarly, some features are influenced by the same functionality being present in a competing product. Both cases, in effect, represent requirements that are implicitly validated by the market.

However, most other requirements, including those originating with user contributions, are informally validated by consensus of the core developers.

Since web browsers are a ubiquitous tool, one would expect that the majority of Firefox developers are also Firefox users with needs that are the same as the larger user community, and so this process of informal validation could be expected to work well. This may not be the case for other open source products with more focused user communities, such as ERP or health care applications.

4 Related Work

Studies of open source software development projects address a wide range of topics from economics [55] to maintainability [47].

A number of case studies have examined open source development processes, including those employed by Apache and Mozilla [29, 44, 46]. In particular, Reis and de Mattos Fortes, in their study of Mozilla development processes, report that high level requirements are specified by the Mozilla Foundation management, but all development on the Mozilla code base originates with a “bug” report, which might be submitted by another developer, tester, or end user [44]. These reports may document some product failure, or a request for enhancement.

Feller and Fitzgerald note that users are a “critical feature” [11, 10] of open source projects, as the source of new requirements. Scacchi has made several studies of requirements acquisition in open source software development; he observes that requirements “emerge” from on-line discussions which are usually open forums, rather than through traditional requirements elicitation processes, but that this emergent process, though less formal, is also effective [45, 46]. He also notes that requirements are “asserted” after the fact; other researchers have echoed this observation. In particular, German reports a similar situation in the Gnome project [13]. This seems to contradict conventional understanding that cites failure to understand requirements as a major source of software project failure.

But Trudelle observes, in his discussion of lessons learned from experience working on Mozilla, that this approach led to rework of some of the Mozilla implementation in response to user-submitted bug reports; his view is that this rework could have been avoided with traditional up-front requirements analysis and design activities [54]. Henderson echoes this view, claiming that open source projects do not employ “requirements elicitation,” but that this could (and should) be easily added to open source processes [20]. Further, Nichols and Twidale observe that usability requirements are not captured well by open source projects, due to the mismatch between developers and users; their view is that the open source approach of “coding as early as possible” violates “good interface design [37].”

These observations run counter to the prevailing open source view that de-emphasizes formal design and requirements gathering, yet also hint at the possible evolution of

mature projects like Firefox. On the one hand, Trudelle's view – that open source software projects need an overarching UI design and design function – seems to contradict the current success of Firefox, which is widely recognized as among the most innovative web browsers. In particular, Nichols and Twidale's assertion that “commercial software establishes the state of the art” [37] seems to be contradicted by Opera and Firefox, both of which included UI features (tabbed browsing, for example) well before Internet Explorer. Yet, the results of this study of Firefox 2 show that some formal user experience design is being undertaken; and, at least three features did follow the lead of competing commercial products.

5 Conclusions

Most Firefox requirements are asserted by developers. The Firefox project does, however, occasionally resort to more rigorous requirements validation, as in the case of the “Visual Refresh” where usability studies were conducted at Google on behalf of the Firefox effort. This confirms the views of Trudelle [54] and Henderson [20], that open source software projects could benefit from more formal requirements analysis.

Firefox is not a “pure” open source project in the spirit of Raymond's community of volunteers scratching an “itch [43].” The Firefox project incorporates some aspects of “traditional” software engineering such as requirements documents and usability studies; it also inherits face-to-face meetings, paid developers, and a formal corporate infrastructure from the Mozilla Foundation. At the same time, the Firefox project has the key aspects of an open source project that distinguish this kind of software development from conventional commercial development: transparency through publicly accessible documents and discussion forums, and direct participation by enthusiastic users.

This philosophy seems to be summed up nicely by the following comment to a usability issue in the Mozilla issue database:

As an average user I would think: how many clicks does it need to get there?
 But there are many different average users :)
 What people really find convenient you can only know when you make a test browser with the new feature, let a thousand users use it for some weeks and then ask the right questions [23].

References

- [1] alanjstr: Bug 231062 provide Firefox MSI package. https://bugzilla.mozilla.org/show_bug.cgi?id=231062 (2004). Issue posted to Firefox issue database, accessed December 12, 2007.
- [2] aldo-public: Comment on bug 270541. https://bugzilla.mozilla.org/show_bug.cgi?id=270541c3 (2005). Comment on request for enhancement posted to Mozilla issue database, accessed October 29, 2007.

- [3] Beltzner, M.: Firefox/feature brainstorming. http://wiki.mozilla.org/index.php?title=Firefox/Feature_Brainstorming&oldid=16495 (2006). Entry in Mozilla Wiki, accessed December 4, 2007.
- [4] Burnett, G.: Firefox 1.5 session saver extension. <http://www.yista.com/gburnett/firefox-15-session-saver-extension/> (2005). Entry posted to the *Yah, I Saw That Already* web log, accessed December 15, 2007.
- [5] Cassin, R.: Bug 56301 (spellchecker) connect a spellchecker engine for Mozilla. https://bugzilla.mozilla.org/show_bug.cgi?id=56301 (2000). Issue posted to the Mozilla issue database describing a need for spell checking in Mozilla Composer, accessed December 14, 2007.
- [6] Champeon, S.: JavaScript: How did we get here? http://www.oreillynet.com/pub/a/javascript/2001/04/06/js_history.html (2001). Web page, accessed December 18, 2007.
- [7] Delahaye, S.: Firebird installer. <http://seb.mozdev.org/firebird/> (2003). Web page, accessed December 12, 2007.
- [8] Dotzler, A.: Reply to 'complete support of XP visual style?'. <http://forums.mozillazine.org/viewtopic.php?p=2668> (2002). Reply to comment posted to Mozilla Themes Development forum, accessed October 3, 2007.
- [9] Epperson, B.: Bug 16409 invoke spell check in browser window (multiple form fields). https://bugzilla.mozilla.org/show_bug.cgi?id=16409 (1999). Issue posted to the Mozilla issue database describing a need for spell checking in the Mozilla web browser, accessed December 14, 2007.
- [10] ExNihilo: Complete support of XP visual style? <http://forums.mozillazine.org/viewtopic.php?t=452> (2002). Comment posted to Mozilla Themes Development forum, accessed October 3, 2007.
- [11] Feller, J., Fitzgerald, B.: A framework analysis of the open source software development paradigm. In: Proceedings of the International Conference on Information Systems, pp. 58–69. Association for Information Systems (2000)
- [12] Ganesh, V.: Top 10 usability features I would like in a browser. <http://geekswithblogs.net/vganesh/archive/2005/09/22/54669.aspx> (2005). Web log entry, accessed December 18, 2007.
- [13] German, D.M.: GNOME, a case of open source global software development. In: Proceedings of the 6th International Workshop on Global Software Development. Portland, OR USA (2003)
- [14] Goodger, B.: Bug 308396 - UE fixes for tabbed browsing. https://bugzilla.mozilla.org/show_bug.cgi?id=308396 (2005). Issue posted to Firefox issue database, accessed December 5, 2007.
- [15] Goodger, B.: Firefox:extension manager UI. http://wiki.mozilla.org/Firefox:Extension_Manager_UI (2005). Entry posted to Mozilla Wiki describing problems and possible solutions with the Firefox extension manager user interface, accessed November 20, 2007.
- [16] Goodger, B.: Link targeting. http://wiki.mozilla.org/Link_Targeting (2006). Entry in Mozilla Wiki, accessed December 4, 2007.

- [17] Goodger, B.: Phishing protection. http://wiki.mozilla.org/Phishing_Protection (2006). Mozilla wiki entry describing history and design of Firefox Phishing Protection feature.
- [18] Google, Inc.: Google toolbar for Firefox. <http://www.google.com/tools/firefox/toolbar/FT3/intl/en/> (2007). Web page for Google Toolbar, accessed December 12, 2007.
- [19] Hamm, S.: A Firefox in IE's henhouse. *Business Week* (2004)
- [20] Henderson, L.G.R.: Requirements elicitation in open-source programs. *CrossTalk - The Journal of Defense Software Engineering* **13**(7), 28–30 (2000). <http://www.stsc.hill.af.mil/crosstalk/2000/07/henderson.html>
- [21] Hiroshi, H.: Tabbrowser extensions. http://piro.sakura.ne.jp/xul/tabextensions/old_index.html.en (2007). Web home page for (now obsolete) Tabbedbrowser Extensions, accessed December 15, 2007.
- [22] jacob667: Using the RSS reader panel [tip 28]. <http://jacob667.livejournal.com/9948.html> (2004). Entry posted to “Jacob’s Mozilla Tips” web log, accessed October 25, 2007.
- [23] Klassen, R.: Comment on bug 335781. https://bugzilla.mozilla.org/show_bug.cgi?id=335781#c2 (2006). Comment to issue posted to Mozilla issue database, accessed November 19, 2007.
- [24] Kottke, J.: Why are Safari and Sherlock two different applications? <http://www.kottke.org/03/01/safari-sherlock-different> (2003). Entry posted to *kottke.org* web log, accessed November 12, 2007.
- [25] Lakhani, K.R., Wolf, R.G.: Perspectives on Free and Open Source Software, chap. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. MIT Press (2005)
- [26] Melez, M.: Microsummaries in Firefox 2.0. http://groups.google.com/group/mozilla.dev.apps.firefox/tree/browse_frm/thread/e6b47d5f7af7d77d/670a06e452b63b9d?rnum=1&_done=%2Fgroup%2Fmozilla.dev.apps.firefox%2Fbrowse_frm%2Fthread%2Fe6b47d5f7af7d77d%2F670a06e452b63b9d%3F#doc_670a06e452b63b9d (2006). Post to mozilla.dev.apps.firefox discussion group, accessed December 10, 2007.
- [27] Melez, M.: son of live bookmarks. <http://www.melez.com/mykzilla/2006/01/son-of-live-bookmarks.html> (2006). Web log entry, accessed December 10, 2007.
- [28] Milford, D.: Bug 270541 - support drag-and-drop of RSS feed (icon) from Firefox. https://bugzilla.mozilla.org/show_bug.cgi?id=270541 (2004). Request for enhancement posted to Mozilla issue database, accessed October 29, 2007.
- [29] Mockus, A., Fielding, R.T., Herbsleb, J.: A case study of open source software development: The Apache server. In: Proceedings of the 22nd International Conference on Software Engineering, pp. 263–272. Limerick, Ireland (2000)

- [30] Mozilla Corporation: Firefox 2. <http://www.mozilla.com/en-US/firefox/> (2007). Web page about the Firefox web browser, accessed December 12, 2007.
- [31] Mozilla Development Team: Firefox 2 release notes. <http://en.www.mozilla.com/en/firefox/2.0/releasenotes/> (2007). Web page, accessed October 26, 2007.
- [32] Mozilla Foundation: Firefox2/HistoricalSchedule. <http://wiki.mozilla.org/Firefox2/HistoricalSchedule> (2006). Web page describing the Firefox 2 release schedule, accessed December 12, 2007.
- [33] Mozilla Foundation: Firefox2/PRD. <http://wiki.mozilla.org/Firefox2/PRD> (2006). Web page containing Mozilla Firefox 2 Product Requirements Document, accessed August 24, 2007
- [34] Mozilla Foundation: Firefox2/Requirements. <http://wiki.mozilla.org/Firefox2/Requirements> (2006). Entry in Mozilla Wiki describing Firefox 2 requirements.
- [35] Mozilla Foundation: Firefox2/StatusMeetings. <http://wiki.mozilla.org/Firefox2/StatusMeetings> (2006). Web page indexing Mozilla Firefox 2 status meeting notes, accessed October 2, 2007.
- [36] Mycroft Project: Mycroft project: Sherlock & OpenSearch search engine plugins. <http://mycroft.mozdev.org/index.html> (2007). Web page describing the Mycroft search engine plugin collection project, accessed November 4, 2007. The “Mycroft” name comes from the name of Sherlock Holmes’s brother, Mycroft, from the novels of Arthur Conan Doyle.
- [37] Nichols, D.M., Twidale, M.B.: The usability of open source software. *First Monday* **8**(1) (2003)
- [38] Nicholson, G.: Bug 238212 Firefox should have a net installer, like Seamonkey. https://bugzilla.mozilla.org/show_bug.cgi?id=238212 (2004). Issue posted to Firefox issue database, accessed December 11, 2007.
- [39] Nicolas: Bug 285848 - extension manager should be able to manage the language of the extensions. https://bugzilla.mozilla.org/show_bug.cgi?id=285848 (2005). Request for enhancement posted to Firefox issue database, accessed November 21, 2007.
- [40] Opera Software ASA: Changelog for Opera 6.1 beta 1 for Linux. <http://www.opera.com/docs/changelogs/linux/610b1/index.dml> (2001). Web page, accessed November 5, 2007.
- [41] Ortyl, P.: Bug 282579 - implement <svg:textPath>. https://bugzilla.mozilla.org/show_bug.cgi?id=282579 (2005). Issue database entry describing Firefox deficiency handling <svg:textPath> tag.
- [42] van Rantwijk, H.J.: MultiZilla’s home page. <http://multizilla.mozdev.org> (2006). Home page for the MultiZilla project, cited September 6, 2006.
- [43] Raymond, E.S.: The cathedral and the bazaar. In: *The Cathedral and the Bazaar*. O’Reilly and Associates (1999)

- [44] Reis, C.R., de Mattos Fortes, R.P.: An overview of the software engineering process in the Mozilla project. In: Proceedings of the Open Source Software Development Workshop. Newcastle upon Tyne, UK (2002)
- [45] Scacchi, W.: Understanding the requirements for developing open source software systems. *IEE Proceedings – Software* **149**(1), 24–39 (2002)
- [46] Scacchi, W.: Free and open source development practices in the game community. *IEEE Software* pp. 59–66 (2004)
- [47] Schach, S.R., Jin, B., Wright, D.R., Heller, G.Z., Offut, A.J.: Maintainability of the Linux kernel. *IEE Proceedings – Software* **149**(1) (2002)
- [48] Schneider, F.: Bug 329292 - add SafeBrowsing anti-phishing extension to trunk for evaluation. https://bugzilla.mozilla.org/show_bug.cgi?id=329292 (2006). Request for enhancement posted to Mozilla issue database, asserting need to integrate Safe Browsing into Firefox core.
- [49] Schultz, K.: Comment on bug 329292. https://bugzilla.mozilla.org/show_bug.cgi?id=329292#c9 (2006). Comment #9 posted to Mozilla issue database discussion of Bug 329292.
- [50] Sellers, D.: Better searching with Sherlock 2. <http://www.computeruser.com/articles/1906,5,18,1,0601,00.html> (2000). Article posted to the Computer User web site, access November 12, 2007.
- [51] Strong, R.: SpellBound - release notes. <http://spellbound.sourceforge.net/relnotes> (2005). Web page containing release history for the SpellBound extension to Firefox, accessed December 14, 2007.
- [52] Strong, R.: Bug 326580 - Firefox 2.0 Windows installer. https://bugzilla.mozilla.org/show_bug.cgi?id=326580 (2006). Issue posted to Firefox issue database, accessed December 11, 2007.
- [53] Torisugari: Spell checker for Firebird. <http://forums.mozillazine.org/viewtopic.php?t=34799&postdays=0&postorder=asc&postsperpage=15&start=0> (2003). Entry in MozillaZine “Extension Development” discussion forum describing initial port of Thunderbird spell checking mechanism to Firebird v. 0.7, accessed December 14, 2007.
- [54] Trudelle, P.: Shall we dance? Ten lessons learned from Netscape’s flirtation with open source UI development. Tech. rep., Mozilla.org (2002). URL `\url{http://www.iol.ie/~calum/chi2002/peter_trudelle.txt}`. Presented at the Open Source Meets Usability Workshop, Conference on Human Factors in Computer Systems (CHI 2002), Minneapolis, MN. Accessed December 28, 2006.
- [55] Wheeler, D.A.: Why open source software / free software (OSS/FS, FLOSS, or FOSS)? Look at the numbers! Technical report, dwheeler.com (2005). URL http://www.dwheeler.com/oss_fs_why.html
- [56] Zotero Project: Zotero - the next generation research tool. <http://www.zotero.org/> (2007). Home page of the Zotero Project, accessed December 15, 2007.

Analysis of Coordination Between Developers and Users in the Apache Community

Yasutaka Kamei¹, Shinsuke Matsumoto¹, Hirotaka Maeshima¹,
Yoji Onishi¹, Masao Ohira¹, and Ken-ichi Matsumoto¹

¹ Graduate School of Information Science, Nara Institute of Science and
Technology 8916-5 Takayama, Ikoma, Nara 630-0192, Japan
{yasuta-k,shinsuke-m,hirotaka-m,yoji-o,masao,matumoto}@is.naist.jp

Abstract. Coordination is one of the keys for the success of open source software (OSS) communities because geographically distributed members need to collaborate on their work using communication tools (e.g., mailing lists, bulletin board systems, bug tracking systems, and so on). In this paper, we investigated the informal social structure among developers and users by analyzing two mailing lists of developers and users in the Apache community based on betweenness centrality, one centrality measure proposed by Freeman. From the analysis results, we found that (1) participants with high betweenness coordinated activities between developers and users and (2) some participants have been functioning as coordinators in the community for a long time.

1 Introduction

Today, open source software (OSS) is widely used by both individuals and users in administrative agencies and educational institutions, since many OSS products with high functionality and quality are free. Generally, OSS is developed in OSS communities on the WWW, where developers and users discuss, share, and realize diverse, innovative ideas on OSS products under development using such electronic media as mailing lists and bulletin board systems [3].

However, the community-based development process occasionally bogs down or remains stagnant due to such social factors as dissension among developers, downturns in user demand, emergence of a superior OSS, and so forth. The cessation of OSS development means end-users cannot obtain continual, sufficient support from the community for bug fixes and functional enhancement. Therefore, system administrators in an organization need to carefully decide whether they should adopt OSS as an alternative for existing commercial software.

As OSS systems become critical information infrastructure in our society, many researchers and practitioners are very interested in understanding *why OSS communities are sometimes very successful and vice versa, how participants in OSS communities can create sustainable communities, what can be done to anticipate whether OSS communities will maintain their activities in the future* and so forth. Recent studies

Please use the following format when citing this chapter:

Kamei, Y., Matsumoto, S., Maeshima, H., Onishi, Y., Ohira, M. and Matsumoto, K., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 81–92.

have tried to reveal the process of OSS development and success factors in OSS communities [7, 9].

For understanding the success factors of OSS communities, for instance, Howison et al. analyzed the informal social structure constructed from communication patterns among developers in OSS communities [6]. They found that a long-lived community does not dynamically change the structure of communications among developers [2, 6]. Raymond pointed out that user feedback to OSS products is also an important success factor [10]. Because such feedback from users as requests for new features and bug reports indicates the demand for their products, responding to such feedback allows developers to address social needs. That is, users play an important role in motivating developers to continuous OSS development.

As Raymond suggested, we also believe that feedback from users is a crucial success factor of OSS development. From this point of view, a prior study [6], which attempted to reveal the informal social structure in OSS communities, might be insufficient to understand the relationships among developers and users. In this paper, we investigate the informal social structure among developers and users, using the history data of communications in OSS communities (e.g., mailing lists).

The rest of our paper is organized as follows. Section 2 introduces related work. Section 3 explains the informal social structure in OSS communities, and Section 4 describes the details of our analysis method. Section 5 provides our case study, and Section 6 discusses the analysis results. Section 7 summarizes our paper and presents some future topics of research.

2 Related Work

Many studies on software development in OSS communities have been reported. Mockus et al. [9] investigated assumptions of OSS development using CVS logs and bug reports and revealed that a mere 4% of Apache developers contributed to 88% of the added lines of code and 66% of the fixed defects. [5] also reported similar results about community-driven OSS development. Although [5, 9] introduced the reality of OSS development from the perspective of software production by OSS developers, they did not suggest interactions among OSS developers (how they collaborate with each other and coordinate their activities). In contrast with these studies, Jensen et al. [7] illustrated role migration and advancement processes in OSS communities including the Apache community, based on qualitative and ethnographic methods. In this paper, we also analyze an aspect of OSS development processes in Apache communities by focusing on coordination processes in OSS development.

Studies have already reported the coordination process in OSS communities. Yamauchi et al. described how geographically distributed developers coordinated their activities and how electronic media were used in the coordination by analyzing the mailing lists of FreeBSD Newconfig and GNU GCC community [11]. Howison et al. analyzed the outdegree centrality of developers in five OSS communities based on bug report data in a bug tracking system [6]. [6] showed that most communities had

a single core developer for long periods of time and that larger communities do not change core developers compared with smaller communities. Bird et al. examined the correlation between centralities and Apache developer contributions from developer mailing lists and the changed history of source code [1]. The analysis results indicated that developers communicating with many other developers contributed more to source code changes. The above three studies clarified the coordination and collaboration processes in OSS communities by focusing on developer activities. In this paper, we analyze the coordination between developers and users based on the above described reasons.

3 Informal Social Structure in OSS Communities

General OSS communities form a kind of online community in which participants are geographically distributed and communicate with each other through electronic media such as mailing lists and bulletin board systems (BBS). They can freely discuss ideas and exchange information on an equal footing without the influence of seniority or social position. OSS communities often provide participants with multiple mailing lists and/or BBS because diverse topics are discussed in the community, depending on the roles of participants. Most OSS communities have several subgroups (e.g., developers using the development mailing list) to encourage specific discussions.

The informal social structure with the above characteristics is represented as a communication network in Figure 1. Here the network is defined by regarding each node as a sender of a message and an edge as a sender-receiver relationship. For instance, if participant A sends message (M) to BBS and then participant B replies to M, A and B are the sender and receiver, respectively, creating a sender-receiver relationship. Based on the individual roles and purposes of the participants in OSS communities, each participant joins one or more subgroups to communicate with other participants. Below, we call participants who only join discussions in a developer group P_{dev} , participants who only have discussions in user group P_{user} , and participants who join discussions in both the developer and user groups $P_{d \cap u}$.

In OSS communities, discussions in the developer and user groups are independent of each other by nature. Therefore, the community requires people who moderately coordinate the activities to maintain their momentum in the OSS communities. We consider that the existence of a person who coordinates the activities between developers and users (e.g., center-black node in Figure 1) has important implications for OSS community activities as a whole.

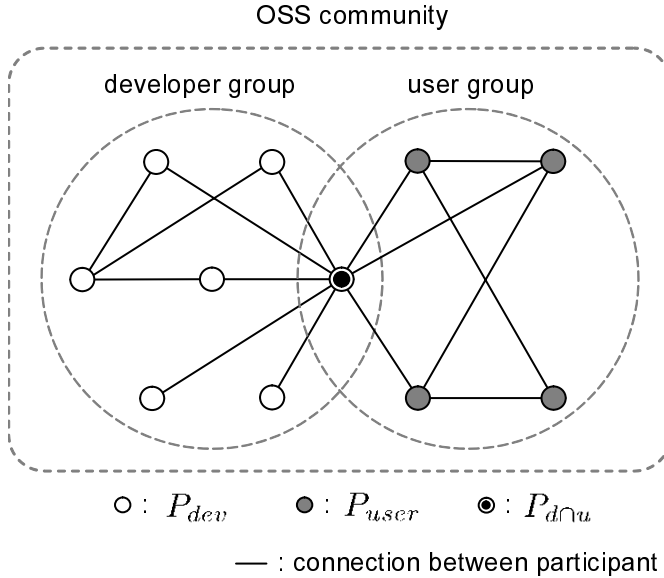


Fig. 1. Informal social structure of OSS community

4 Analysis

4.1 Analysis of coordinators

In this paper, we suppose $P_{d \cap u}$, which was discussed in both the developer and user groups, as a coordinator who arranges interactions between developers and users. We focus on a coordinator as a person who actively mediates between them. $P_{d \cap u}$ with a high degree of intermediation tightly connects developers and users and coordinates activities between developers and users. Howison et al. applied the outdegree centrality measure to the developer group in Apache to identify developers who provide a variety of information with other developers [6]. In this paper, we analyze and identify the developers who strongly connect developers and users in Apache. We use betweenness centrality, one of the centrality measures proposed by Freeman [4], as the degree of intermediation between both groups. Our analysis is composed of the following.

Visualizing the informal social structure

Visualizing the informal social structure to understand the whole picture of the structure.

Calculating the betweenness centrality

Identifying $P_{d \cap u}$ with a high degree of intermediation between developers and users by calculating the betweenness centrality for each actor in the network.

Reading contents of messages

Reading the contents of messages posted by $P_{d \cap u}$ that show high betweenness to confirm whether $P_{d \cap u}$ actually coordinates activities between both groups.

4.2 Betweenness centrality

The following describes the betweenness centrality measure proposed by Freeman [4]. Betweenness Centrality $C_{betweenness}(v_i)$ is formulated as below.

$$C_{betweenness}(v_i) = \frac{\sum^n \sum_{j < k}^n p_{jk}(v_i)}{\sum^n \sum_{j < k}^n p_{jk}}, \quad (1)$$

where n shows the number of nodes in the network, p_{jk} shows the shortest path from v_j to v_k , $\sum^n \sum_{j < k}^n p_{jk}$ shows the number of paths from v_j to v_k , and $\sum^n \sum_{j < k}^n p_{jk}(v_i)$ shows the number of paths from v_j to v_k including v_i .

$C_{betweenness}(v_i)$ takes a value from 0 to 1, and a node with a higher betweenness indicates a node that acts as an intermedior in the network. In OSS communities, we assume a node with high betweenness centrality is a participant who strongly connects other participants. Thus, if a participant with high betweenness centrality suddenly leaves the project, other participants might have trouble communicating with each other.

5 Case Study

This section describes our case study of the Apache HTTP Server community that has been developing a web server software product with the biggest market share.

5.1 Data source

In the case study, we used the following archival data sources.

Developer mailing lists

Anyone interested in working on Apache development can join this mailing list that has been archived monthly since March 1995. It is mainly used for discussions on the development of Apache, including technical topics and bug fixes.

User mailing lists

Anyone who needs support to use Apache can join this mailing list that has been archived monthly since November 2001. It is mainly used for user support, including questions and answers about Apache installation and reports about error comments.

We analyzed the above two mailing lists for three months before and after the release of the latest major version of Apache (2.2.0). We consider the person who only sends e-mails to the developer mailing lists as P_{dev} , the person who only sends emails to the user mailing lists only as P_{user} and the person who sends emails to both of them as $P_{d \cap u}$.

5.2 Data cleaning

Since some participants in the mailing lists may have several email addresses, we need to clean the data to identify such participants. We identified the same person in the mailing lists based on the following steps.

- Step 1. *A sender of messages with the same email address and different "Name"*: If the same email addresses are used by different "Name" senders such as "Alice" and "Bobby," we consider them the same person.
- Step 2. *A sender of messages with the same name at "From" and partially the same address before at mark (@)*: If two email addresses such as `chris@domain1.com` and `chris@domain2.com` are used by the same "Name" sender such as "Chris," we consider them the same person.
- Step 3. *A sender of messages with the same name at "From" and different email addresses*: If two email addresses such as `daniel@domain3.org` and `johnson@domain4.edu` are used by the same "Name" sender such as "Daniel," we judge they are used by the same person after confirming the body of the messages (e.g., messages with the same signature).

In cleaning the data, we found 740 unique senders by applying step 1 to the data, 683 unique senders by applying step 2 to the data after step 1, and 678 unique senders by applying step 3 to the data after step 2.

5.3 Analysis results

Informal social structure We visualized the informal social structure in the Apache community based on the definition described in Section 3 to understand the whole picture of the community structure.

Figure 2 represents the network of the Apache community for the period when version 2.2.0 was released. In Figure 2, developers P_{dev} , users P_{user} , and $P_{d \cap u}$ are positioned respectively at the left, right, and center. There are $P_{d \cap u}$ with many edges between both users and developers.

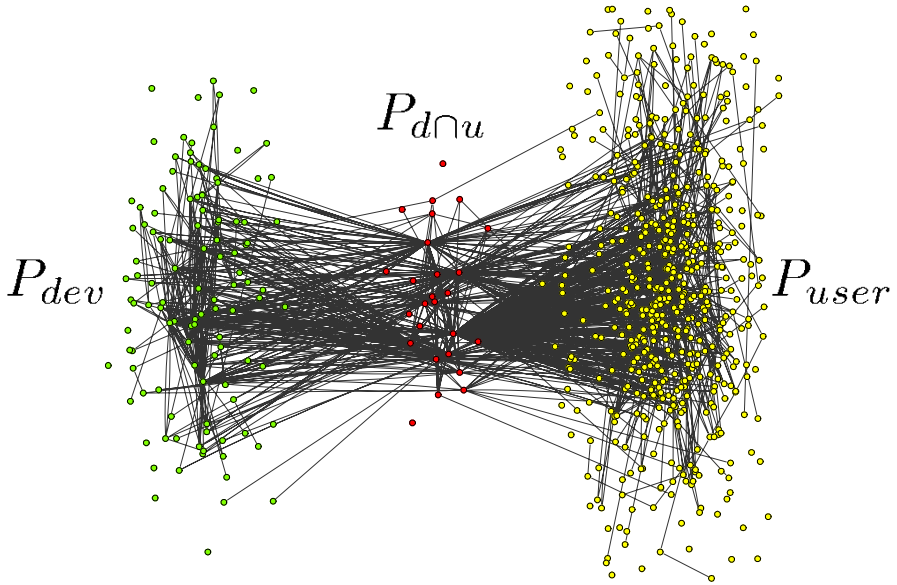


Fig. 2. Informal social structure in Apache community

Betweenness centrality We calculated the betweenness centrality of each P_{dru} in the network to identify the P_{dru} with high degree of intermediation between P_{dev} and P_{user} . Figure 3 shows the network of the top 5 P_{dru} of betweenness centrality. Comparing the two networks in Figures 2 and 3, we can see the top 5 P_{dru} intermediates between more than half of the users and developers. Counting the number of edges of the top 5 P_{dru} , they communicated with 55 of 112 P_{dev} in the developer mailing list and with 249 of 540 P_{user} in the user mailing list.

Table 1. Statistics of top 5 P_{dru}

	$P1_{dru}$	$P2_{dru}$	$P3_{dru}$	$P4_{dru}$	$P5_{dru}$	median
Betweenness	0.179	0.044	0.043	0.022	0.019	0.001
Num. of emails	592	193	261	127	62	17
Num. of degrees with developers	15	29	33	18	11	2
Num. of degrees with users	189	31	28	19	21	1

Table 1 indicates the statistical values of the top 5 P_{dru} . $P1_{dru}$ with the highest betweenness has an extremely large number of degrees with developers (P_{dev}), compared with the left P_{dru} . The betweenness centrality of the top 5 P_{dru} is 10 times

higher than the median, meaning they act as intermediates between many developers and users.

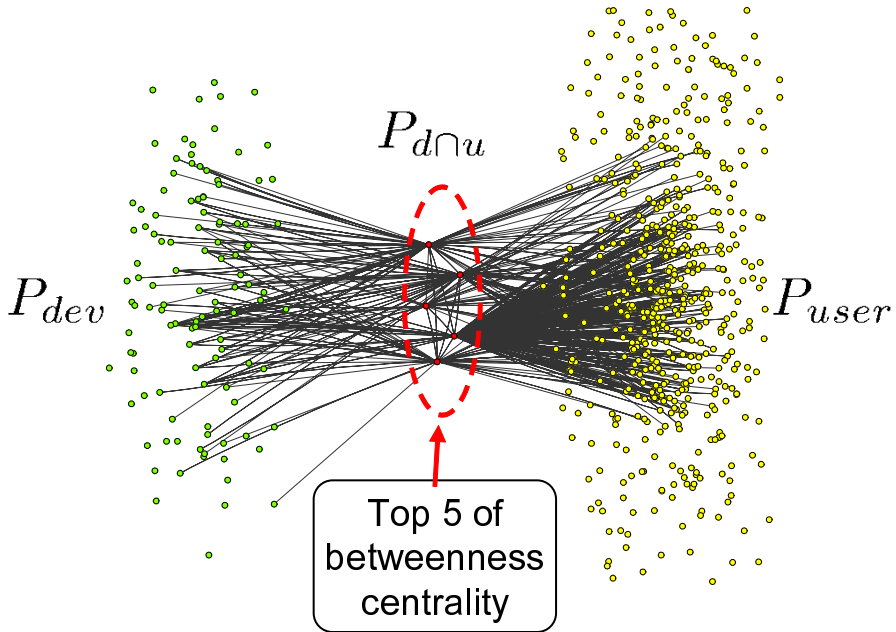


Fig. 3. Informal social structure of top 5 $P_{d \cap u}$

Content of messages We checked the contents of messages posted by the top 5 $P_{d \cap u}$ to confirm whether they actually coordinated activities between both P_{dev} and P_{user} . Five experimenters judged whether the messages related to the coordination actions of $P_{d \cap u}$. If more than three experimenters judged that a message implied coordination actions, we decided the message is related to coordination actions.

From experimenter reviews of the content of messages, we found that the top 5 $P_{d \cap u}$ with high betweenness centrality coordinated activities between developers and users. Due to space limitations in this paper, we can only show part of the reviews in Tables 2, 3, and 4.

Table 2 shows a kind of coordinative action, which is a guide of discussions (users \rightarrow developers). If developer-related topics were discussed in a user group and $P_{d \cap u}$ guided them to be discussed in a developer group, we considered such action of $P_{d \cap u}$ coordination. For instance, as in Table 2, this should be discussed in a developer group, because the topic concerns implementation. In this situation, coordination is required to guide the discussions to an appropriate place (developer group).

Table 2. Guide of discussions: users→developers

sender	dialog	notes
user	<i>I implemented a module of Apache in a Windows environment. I want to run it on Linux.</i>	The user is asking a user group about the module development of Apache.
$P1_{d\rightarrow u}$	<i>This should be discussed at developer ML because it is an implementation matter.</i>	$P1_{d\rightarrow u}$ is guiding the user to the developer group.

Table 3 shows a kind of coordinative action, which is information transfer (users→developers). If $P_{d\rightarrow u}$ transferred information to a developer group that only users had, we considered such action of $P_{d\rightarrow u}$ coordination. For instance, as in Figure 3, requests for new features and product evaluations should be conveyed to developers for further development. In this situation, coordination is required to provide user feedback to a developer group.

Table 3. Information transfer: users→developers

sender	dialog	notes
developer	<i>Some versions do not seem very popular.</i>	The developer is suggesting in the developer group that some versions of Apache should be stopped from being made public because they are not popular.
$P3_{d\rightarrow u}$	<i>Since I have personally received emails from users regarding their versions, I think the versions are still popular.</i>	$P3_{d\rightarrow u}$ is telling the developers that the versions mentioned by the developers are still popular. He is motivating the developers to continue Apache development by describing its popularity among users.

Table 4 shows a kind of coordinative action, which is a request for participation (developers→users). The shortage of members in one group means that members not only in the group but also the developer group face bigger burdens. For instance, the shortage of testers, as in Table 4, imposes not only on current testers but also on developers because they have to develop and test the software. In this situation, coordination is required to ask users to participate in a group with few members.

Table 4. Request for participation: developers→users

sender	dialog	notes
$P3_{d\rightarrow u}$	<i>It is short of testers for some minor OS. We need your contributions as testers.</i>	Due to the shortage of testers for some versions of Apache compiled for some minor OS, $P3_{d\rightarrow u}$ asks the user group to participate in the tester group. Such coordination would contribute to the development by reducing the burden of developers.

6 Discussion

From the analysis results focusing on the betweenness centrality, we confirmed the existence of coordinators (the top 5 P_{dru} of the betweenness) who support the activities of developers and users. However, since the analysis target in Section 5.3 was only the data for the three months before and after the latest major version of Apache (2.2.0) was released, it remains unclear whether the top 5 P_{dru} only supported the activities of developers and users for the analysis period or also for other periods. In this section, we analyze the transition of the betweenness centrality of the top 5 P_{dru} from November 2001 to September 2006.

Figure 4 shows the change in the ranking of the top 5 P_{dru} over time. The x- and y-axes indicate the time and the betweenness centrality, respectively. We calculated the betweenness centrality using the sliding time method [8]. Here, centrality is calculated by sliding the three month analysis window month-by-month.

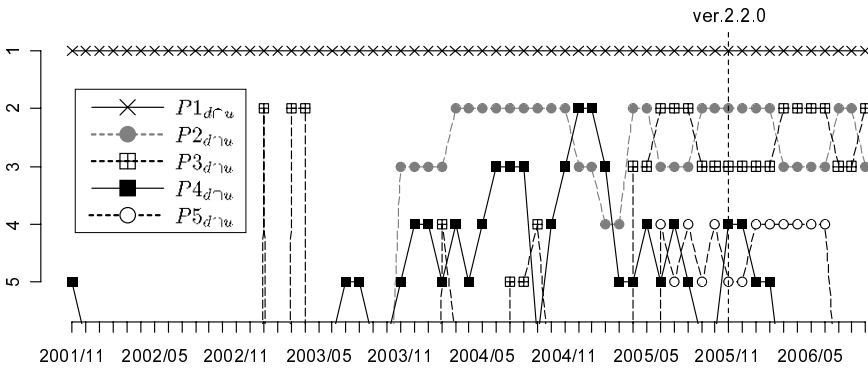


Fig. 3. Informal social structure of top 5 P_{dru}

Figure 4 shows the existence of particular P_{dru} who have been intermediating between developers and users for a long time. The betweenness centrality of $P1_{dru}$ was the highest for the whole period. Though we did not check all messages posted by $P1_{dru}$, we believe that coordinators with consistently higher betweenness such as $P1_{dru}$ continues coordinating activities between developers and users. In the Linux community, one of the most successful communities, Linus Torvalds, coordinator of the Linux Kernel community, has been contributing to its development since its start in 1991. One of the success factors of the OSS communities is the existence of coordinators who facilitate and coordinate activities among the members of OSS communities.

7 Conclusion and Future Work

In this paper, we investigated the informal social structure among developers and users using two mailing lists of developers and users in the Apache community. The following are the findings of our case study.

- participants with high betweenness coordinated activities between developers, and
- some participants have been functioning as coordinators in the community for a long time.

Here, note that our analysis results are applicable to the data from the mailing lists used in the Apache community. We need to investigate other datasets (e.g., history data in bug tracking systems) and other communities to increase the appropriateness of our results. Furthermore, the data cleaning described in Section might be imperfect because we did not find any message senders who used different names at “From” and different email addresses. We need to increase the sophistication of the data cleaning method in the near future.

Acknowledgments

This research was conducted as part of the EASE project in the Comprehensive Development of e-Society Foundation Software program, and the Stage Project, the Development of Next Generation IT Infrastructure, and Grant-in-aid for Scientific Research (B) 17300007, 2007 and for Young Scientists (B), 17700111, 2007, by the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

1. Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR'06), pp. 137-143, 2006.
2. Kevin Crowston and James Howison. The social structure of free and open source software development. *First Monday*, Vol. 10, No. 2, 2005.
3. Joseph Feller and Brian Fitzgerald. *Understanding Open Source Software Development*. Addison-Wesley, 2002.
4. Linton C. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, Vol. 1, No. 3, pp. 215-239, 1979.
5. Daniel German and Audris Mockus. Automating the measurement of open source projects. In Proceedings of the 3rd Workshop on Open Source Software Engineering, pp. 63-67, Portland, Oregon, 2003.

6. James Howison, Keisuke Inoue, and Kevin Crowston. Social dynamics of free and open source team communications. In Proceedings of the 2nd International Conference on Open Source Systems (OSS'06), pp. 319-330, 2006.
7. Chris Jensen and Walt Scacchi. Role migration and advancement processes in ossd projects: A comparative case study. In Proceedings of the 29th International Conference on Software Engineering (ICSE'07), pp. 364-374, 2007.
8. Takeshi Kakimoto, Yasutaka Kamei, Masao Ohira, and Kenichi Matsumoto. Social network analysis on communications for knowledge collaboration in oss communities. In Proceedings of the International Workshop on Supporting Knowledge Collaboration in Software Development (KCSO'06), pp. 35-41, September 2006.
9. Audris Mockus, Roy T Fielding, and James D Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, Vol. 11, No. 3, pp. 309-346, 2002.
10. Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly and Associates, 1999.
11. Yutaka Yamauchi, Makoto Yokozawa, Takeshi Shinohara, and Toru Ishida. Collaboration with lean media: How open-source software succeeds. In Proceedings of the 2000 Conference on Computer Supported Cooperative Work (CSCW'00), pp. 329-338, 2000.

Lost and Gained in Translation: Adoption of Open Source Software Development at Hewlett-Packard

Catharina Melian¹ and Magnus Mähring²

¹ Stockholm School of Economics, Center for Media and Economic
Psychology, Box 6501, SE-11383 Stockholm, Sweden
catharina.melian@hhs.se

WWW home page: <http://www.sse.edu>

² Stockholm School of Economics, Department of Marketing and
Strategy, Box 6501, SE-11383 Stockholm, Sweden
magnus.mahring@sse.se

WWW home page: <http://www.sse.edu>

Abstract. What happens when an organization form that has emerged in one context is brought into a different context? In this paper, a longitudinal field study approach is used to explore how Hewlett-Packard (HP) molded open source software development (OSSD) into a proprietary software development approach called “Progressive Open Source” (POS). With the help of actor-network theory, we understand this as a process of translation and find that some central characteristics of OSSD were lost in the translation into POS while other characteristics were gained.

1 Introduction

The advent of open source software development (OSSD) has spurred considerable interest not only amongst individual systems developers participating in voluntary and unpaid software development over the Internet, but also increasingly amongst software development organizations seeking to improve the quality and productivity of their products and services [23, 34]. In this paper, we explore what happens when OSSD is brought into a traditional, high-performing, software development organization? With the help of actor-network theory, we explore this process of *translation* [6] in Hewlett-Packard, where key actors shaped and constructed the idea of Progressive Open Source (POS) through interactions and through the meanings that they assigned to their work, interactions and experiences [18]. The purpose of the study is thus to explore the translation of an extra-organizational form of organizing, open source software development, into a corporate context.

Please use the following format when citing this chapter:

Melian, C. and Mähring, M., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 93–104.

2 Background

Typically, open source software development takes place in Internet-based communities of software developers who voluntarily collaborate to develop free software that they or their organizations need [26, 34]. OSSD can be seen as a new way of organizing—different from software development conducted according to the traditional paradigm [24, 28]. While there is a lot of interest in adoption and adaption of OSSD within a corporate context, and development platforms for OSSD are now offered in corporate versions [3], little research has so far been devoted to the emergence and anatomy of “hybrid” organizational forms that combine OSSD practices with corporate requirements and needs [29].

Actor-network theory (ANT) addresses how technology influences and is influenced by social elements in a setting over time [6, 7, 20]. According to ANT, the ideas, values and intentions of social actors become inscribed in a technology through interaction. As a result, intentions become immutable [1, 2, 18]. The process through which this occurs involves creation of networks of aligned interests: actors form alliances, enroll other actors and use non-human actors (artifacts, technologies) to secure their interests. The resulting actor-networks are made up of human and non-human actors [7, 19].

The creation of an actor-network, referred to as translation or “the methods by which an actor enrolls others” [5, p. xvii], results—if successful—in a stable actor-network that encompasses non-human actors. In our case, the focal non-human actor is the idea or technology of open source. ANT predicts that the translation of OSSD into a particular context includes some transformation of the idea/technology; the involved actors become aligned in interests, commitments and obligations and as part of this development the idea/technology is instantiated or situated [32].

In this paper, we employ two supplementing views of translation. Steiner [31] emphasizes the journey (or re-formation) of an idea, while Callon [6] emphasizes the formation of the network of actors that in an interactive process constructs itself and incorporates the idea/technology. With this combined view of translation, the process can be described as follows:

1. *Initial act of trust* [31] and *problematization* [6]: At the beginning of the process, a translator assumes that a particular model or idea (e.g., OSSD) makes sense and that it can be reproduced in a different context [31]. The translator opens up for new ways of thinking, and through social interaction the identities and interests of a set of relevant actors start to emerge [6].
2. *Incursion and extraction* [31] and *interessément* [6]: The translator(s) raids the original idea for sense, and bring back that which makes particular sense for the organization. Relatedly, actors attempt to define “the problem” by manipulating and (re)negotiating relationships between concerned actors.
3. *Incorporation* [31] and *enrolment* [6]: In this stage, actors mold, or give particular form and body, to those aspects of an idea that fit into the organizational texture and the idea becomes incorporated. As this happens, actors come to accept

the designated roles attributed to them and the formation of an actor-network occurs.

4. *Appropriation* [31] and *mobilization* [6]: At this stage, the original idea is transformed and becomes intertwined with the organization. The idea is also appropriated—the new idea has taken over all relevant meaning from the old idea and the original idea has no separate identity anymore. Correspondingly, some actors come to represent others and mechanisms are put in place that stabilize the actor-network and keep actors behaving in accordance with interests. At this point, the actor-network and its ideas, technology, and artifacts have become institutionalized and are no longer seen as controversial.
5. *Reciprocation* [31]: While the actor-network is already at this point stable, the translated idea may at this point come to enhance the original idea. For example, the idea resulting from a translation may, if successfully institutionalized, contribute to enhancing the stature of the original idea.

The idea/technology in focus in this study is OSSD, and the translation studied is the re-formation of OSSD into POS within Hewlett-Packard. The challenge of adapting OSSD in a corporate context [29] can thus be seen as dependent on the molding of the idea of open source into a form that fits the context and the corresponding formation of a stable actor-network that performs the appropriation of the idea and sustains the idea and its use. On the basis of this argument and our brief exposé of ANT, we argue that ANT is well suited to advance our understanding of the challenge of corporate OSSD adoption. However, we have not found any studies of corporate OSSD adoption that employ ANT and only one study of OSSD using ANT outside of the corporate context [33]. This sets the stage for our study.

3 Methodology

In 2001, the first author established initial contact with Hewlett-Packard (HP). An agreement was reached to document HP's efforts to introduce the new software development paradigm within the organization. Access to contacts, data and other resources was provided and the field researcher was assigned an office cubicle at HP labs in Palo Alto, California.

The longitudinal field study [4, 25] encompassed nine months in the field at HP between October 2001 and April 2002, in August 2002, and in May 2003. Time spent in the field included direct observation of work, participation in meetings, participation in workplace conversations and dialogues, and extensive field note writing [8, 9]. The study also encompassed 52 formal interviews (semi-structured interviews averaging two hours that were taped and transcribed), corporate and external documents and other forms of secondary data [27]. The data was subsequently coded using NVivo software for qualitative data analysis.¹ For the purpose of this paper, an

¹ For more information on research methodology and case details, see Melian [22].

actor-network lens was selected to further deepen the emerging understanding of the transformation (or translation) of OSSD in the HP context. In particular, the translation model presented in the previous section was used to structure, present and analyze the case.

4 The Translation of Open Source at Hewlett-Packard

In this section we briefly summarize the chains of events that resulted in the widespread use of progressive open source (POS) within Hewlett-Packard Company in the period from 2001 to 2003. Then as now, HP was an engineering company where product development and hi-tech engineering were high-status activities seen as central to the organization's survival and success.

4.1 Initial Act of Trust and Problematization

At the end of the 1990s, actors in several units within HP were involved in discussions on challenges related to product development. The 1990s had seen extremely rapid development in IT products and services as well as a rapid increase in demand for such products and services. There were recurrent discussions within different software development units in HP on the need to reform the software development process and a number of possible remedies and aspirations were studied. In order to increase cost efficiency and shorten time to market, it was perceived necessary to increase use of third parties (for collaborative and outsourced development) and to conduct around the clock ("24/7") development. In addition, software modularization and extensive, systematic reuse of software components was seen as an approach to avoid "reinventing the wheel" and to increase component compatibility.

A discussion came up about how we need to find a way we can work better with third parties, and there was some bantering about how there is Open Source, but we have security needs that we have to make sure that we don't expose our IP².... — HP Software developer.

Many developers within HP were already tinkering with OSSD in their free time and examples of successful open source projects (Linux, Apache web server) were well-known within HP.

The idea of open source as an idea was thus met with an initial act of trust: actors within HP believed OSSD to offer an attractive approach and solution to perceived problems and challenges [31]. The descriptions and definitions of problems were, from an ANT perspective, not objective depictions of a "real" reality, but rather expressions of interests and interpretations by actors involved in the organizational discourse [7, 19]. The engineering community within HP was dominant in this initial identification of OSSD as a preferred idea for organizational betterment and object of trust.

² Intellectual property.

4.2 Incursion and Extraction, and Intersèment

The chief technology officer (CTO) of the Imaging and Printing business area, Lee Caldwell, initiated a project to address these issues, called CDP (for Collaborative Development Program). Robert (Rob) Miller, a senior scientist at HP, was appointed to lead the project.

Rob was just having the idea that this is just one of his jobs, and he kind of looking around and it was a combination of both Open Source and just the issue around collaboration and being able to tap into the expertise around the business and the company. — HP Software developer.

Miller started to search for successful examples of collaboration between development teams in HP and found the Sirius Firmware Cooperative, a group that had developed new firmware architecture for multifunction peripherals in the early 1990s. Four geographically dispersed teams collaborated using a common work process with interconnected core teams and a well-defined peer review process. In order to learn more about Sirius, Miller contacted Cathy Ammirati, head of the Sirius co-op. Ammirati was subsequently invited to join the CDP effort, while Sirius continued to function as a separate entity. In parallel, people involved in the CDP effort read the open source literature and tapped into the discourse on the potential of corporate adoption of open source present in Silicon Valley at this time.

I started learning about open source, and did the typical thing most people do, read ‘The Cathedral and the Bazaar’,³ started reading up as much as I could about it and said its very fascinating to see how we could bring this into HP, and how it could work inside of HP in such a structured environment of Hewlett-Packard... — Software developer involved in setting up CDP.

While the CDP effort was initiated by the CTO and had strong corporate backing, engineers also maintained and nurtured open source as “their thing”. For example, the adoption of open source in the corporate setting was seen as a way to cope with the organizational “reality” shaped by the administrative/managerial community. In this way, different actors were able to selectively assign meaning to the new idea [31] and inscribe it with their intentions [1, 18].

4.3 Incorporation and Enrolment

By now the approach of CDP was to try to learn and evaluate what open source could mean to HP through experience with real software development tasks. By July 2000, the first tasks were set up in the CDP environment, which was still under development:

...we are driving the car down the road and we are improving it as we go. We’re in there tinkering under the hood, somebody’s hanging off the side fixing the door lock.... — Manager, CDP.

³ See Raymond [26].

Independent of the CDP effort, senior scientist Pankaj Garg at HP Labs had at this time initiated a software development environment, Corporate Source (CS), devised to function as an open source environment restricted to HP employees. Most of the development effort went into CDP as a development platform (including development, storage and communication tools) was purchased. CS was used separately and was continuously compared with CDP. In late 2000, the concept of progressive open source (POS) was put forward as a way to describe open source initiatives within HP. HP's increasing attention to *open source* was also reflected in public speeches by HP's CEO:

The open source movement is natural, inevitable and creates huge benefits. It's part of the next wave of computing, and that will involve participants and users within the industry in open source. — CEO Carly Fiorina, October 2000.

Within CDP, a set of defined benefits were formulated that matched the previously identified challenges for HP software development. These included increased code reuse, improved code quality, faster development, faster debugging, and selected partner collaboration. In addition, rapid re-deployment of key developers between projects was seen as facilitated by the common platform and work process. Leading actors posted internal “tech reports” (later presented at public conference [10, 11]) that outlined the benefits of adopting open source within Hewlett-Packard.

The creation of the “POS” concept can be understood as creation of a vessel that facilitated the extraction of meaning from OSSD and the transfer of relevant meaning from the original to the new concept [31]. Also, this phase sees the introduction of an important non-human actor: the CDP development platform and the key attributes of the platform and related services (such as security, search and navigation, migration, and technical support).

4.4. Appropriation and Mobilization

In late 2001 and early 2002, POS was established as a new and paradigmatically different approach to software development within HP. The different breeds and varieties of “open source” initiatives within HP are now explicitly and systematically described as being part of POS, with the notable exception of Sirius. Over a period of 18 months from the original launch of the CDP development platform, there was considerable growth in the number of developers on the CDP platform. The pilot project had encompassed 600 developers. In April 2001, CDP moved to a production environment and by August there were over 2000 users, ten percent of which outside HP. By January 2002 there were 3500 users and over the next few years, the CDP environment would grow to over 10,000 users.

The diffusion of the CDP platform was supported through increasingly well-defined roles: executive champions (the chief technology officer and the chief information officer), sponsors, R&D change leaders, and information technology staff. The executive champions stressed the need for collaboration between development groups in the organization and applied pressure on software managers to transition to the CDP platform. The sponsors committed development resources and the R&D

change leaders were appointed to enable and support a dialogue of culture change and POS training. The IT staff provided the critical collaboration infrastructure on a 24x7 supported basis.

The standardization of development work through the CDP development platform was a painful process for many development teams, but in most cases teams decided to adopt and adapt to the platform and related work methods. The CDP platform thus constituted a mechanism—inscribed with intentions—that stabilized the actor-network and restricted actors to behave in accordance with inscribed interests [6]. POS and CDP had by this time become dominant concepts to describe activities within HP aiming at open, collaborative development. Open source was no longer an important or frequently used concept. In accordance with Steiner's [31] view of translation, the new idea had taken over all relevant meaning from the old idea and the original idea had no separate identity in this particular context anymore.

As more developers transition into the CDP development environment, the development platform becomes an increasingly important non-human actor: Because of the standardization on CDP, third-party developers collaborating with HP had to transition into CDP. Acceptance of the CDP environment and its rules and roles became a necessary prerequisite for continuing to do business with HP.

However, as the use of CDP increased quite rapidly, scalability problems appeared: there were several breakdowns of the development platform during the growth period during 2002. In spite of the breakdowns, the actor-network had become strong and stable enough to impose itself on a gradually larger percentage of HP developers who are faced with a “take it or leave it” offer of joining CDP. For individual developers, there was a risk that the change in development environment would manifest itself in loss of productivity and motivation.

More profound changes in the software development process also started to appear. The previous common practice of *working* “under the radar” was no longer a preferred or even allowed approach to development work. Instead, all development work needed to be out in the open on the CDP platform. Software developers perceived increased pressure as a result of the open and transparent process: In the POS/CDP environment, every contribution, absence of contribution, mistake, misunderstanding, and spontaneous comment was directly attributable to an identified individual. This was in direct contrast to OSSD, where contributing developers are normally anonymous. While POS technology facilitated openness it also besieged privacy and exposed the individual:

...the fishbowl aspect of feeling like you're being watched with every little thing you do. ... It does affect the way that I post to the forum. — Software developer.

In relation to this, HP managers started to comment to each other that a new “breed” of employee was needed—people that could cope with the “fish bowl” visibility of their actions and not let that hamper their openness or they way they interact with others.

4.5 Reciprocation

Through POS, HP managed to improve the efficiency, speed and quality of software development. Unlike the original idea of open source, the POS reincarnation also adhered to traditional criteria for software development in that development work was successfully conducted within time and budget. In this sense, POS can be seen as contributing to the status of OSSD [31] by demonstrating its superiority as development approach also in the (foreign) corporate context.

5 Discussion

We have followed the OSSD idea/technology as it traveled from its original, extra-organizational context and translated into POS; transformed, yet partly the same [17, 31]. Main differences between open source software development and progressive open source in Hewlett-Packard are found in

POS became different from OSSD in order to survive and prosper; that is, in order to become an integral part of a stable and growing actor-network [6, 18]. In particular, our analysis of the case shows that key elements of the HP organization were perceived to be at odds with OSSD *as organizing*. The formal hierarchy and related reporting and control structure—including financial and other goals per unit and area, individual evaluations and rewards—as well as the HP culture were also seen as dominant and immutable features of the HP organization.

Consequently, the incursion and extraction of meaning [31] from OSSD by HP actors focused on those elements of OSSD that were compatible with key organizational design elements of the existing HP organization. The translation of OSSD into POS was mainly intended to change work practices related to software development work, to extend those work practices to incorporate selected corporate partners in controlled ways, and to allow tinkering with open communities for software development.

In this respect, POS can be considered a success. The goals of 24/7 development, more effective reuse of technology and increased number of parallel development processes were achieved. The spread of POS within HP has also been considerable and the practices developed during the case period are still used. In other words, a stable and growing actor-network that formed by the originators of POS, over 10,000 developers, the idea of POS and the CDP development platform. It can also be argued that the success of POS within HP has enhanced the original idea of OSSD (i.e., reciprocation) by making OSSD a more credible idea/technology for organizational adaption [31].

Table 1. A Comparison of Open Source and Progressive Open Source

Open Source Software Development (OSSD) [Based on 12, 15, 16, 26, 30, 35, 36]	Progressive Open Source (POS)
No formal organization, rather a social phenomenon.	Formal organizational entity, sometimes involving 3 rd party developers.
Gift system [21].	Market system.
Underlying notion: abundance.	Underlying notion: scarcity.
Each gift is part of system of reciprocity. Every gift has to be returned in a perpetual cycle of exchanges.	Calculated gains and losses, utility maximization.
Very open and free flow of information.	Progressively more open – however controlled flow of information between trusted partners internal and sometimes external to the formal organization.
Voluntary teams working on projects.	Designated teams, i.e. employees and/or contractors working on defined projects loyal to a formal organization.
Meritocracy/benevolent dictatorship.	Traditional hierarchical organization.
Sociological phenomenon. Has the potential of bringing together software practitioners regardless of their prior record (employment, education, age, status, etc.). Only the validity of the contribution counts.	Organizational phenomenon. Technology in place that enables collaboration between software engineers employed and/or contracted by the organization and trusted partners.
No time and budget constraints.	Often strict time frames and budget restrictions.
Potentially limitless influx of knowledge and skills – infinite number of developers.	Controlled number of developers. Significantly smaller than Open Source community.
Public good: Non-rival (one person's consumption of the good does not reduce the amount available to another) and non-excludable (excluding others from consuming the public good is difficult or impossible)	Privately owned/controlled intellectual property rights.
Challenge:	Challenge:
How to get people to contribute and avoid free riding.	How to move from a control based organization to an organization based on empowerment and trust.
How to coordinate a large group of individuals that want to contribute.	How to reward collaborative behavior.
Motivation to contribute:	Motivation to contribute:
Identity (honor, reputation, ego-boosting)	Monetary
Reciprocity	Patents
Efficacy: individuals contribute because they have some impact on the system.	Ranking system/reward-system
Art and beauty of solving problems	
Work as vocation	

However, several interesting apparent paradoxes can be observed through comparison between OSSD and POS. POS has not only failed to fulfill some of the promises of the OSSD idea: in some aspects, POS produces the opposite of the original OSSD idea. While OSSD takes place in loosely coupled cyberspace communities [16] with weak ties [14] and without many of the restrictions and requirements of corporate contexts [16], the introduction of POS at HP increased the level of structure and standardization of development work. Standardization included forms of expression, language use, defined layers of openness, and the requirement to safeguard intellectual property also in controlled “open” source projects. In some ways, this was far from the corporate culture of “cowboy development”, while in other ways it further enabled micro-managing by managers.

There were also considerable resemblances between OSSD and POS, such as transparency of the development process, constant up-to-date documentation, the large number of “eyeballs” involved in debugging, and 24/7 development. Together, these similarities suggest that central aspects of the open source development work process remained quite intact through the translation process. On the individual level, however, the differences were fundamental. Whereas OSSD enables flexible participation in development work with limited real life consequences, POS has created quite the opposite work environment. In the organizational context, with long-term social relationships and well-defined hierarchy, POS makes the quality and level of contribution of individual developers highly visible. This visibility shapes an environment where an underperforming employee can neither run, nor hide from scrutiny—a virtual panopticon in which every mistake is likely to get noticed [13]. In addition, the balance between openness and safeguarding of intellectual property places intellectual property decisions with individual developers and makes these decisions a part of daily practice.

6 Conclusions and Implications

Using actor-network theory, this research has studied the appropriation of open source practices in HP as a translation of an organization form from an extra-organizational context to a corporate context. The study shows that in the translation of open source into the corporate context, some fundamental aspects of OSSD are lost, while other fundamental aspects of the new hybrid form of software development—progressive open source—are shaped.

The study strongly suggests that a translation of OSSD into the corporate context may come to yield substantial and lasting benefits to efficiency, speed, quality, flexibility and reusability in software development. However, our study also reveals that while many of the characteristics of the work process may remain intact through the translation, the organizational characteristics of open source in the corporate context, particularly in terms of monitoring and control of the individual, may starkly contrast traditional open source environments. The implications of these findings for stake-

holders in organizations considering OSSD adoption are far-reaching and will hopefully inspire further research.

Acknowledgments

We sincerely thank the people in Hewlett-Packard for generously giving of their time and thoughts, and for their enthusiasm—for their own work as well as for this study.

References

1. Akrich, M. The de-scription of technical objects, in W.E. Bijker and J. Law, ed., *Shaping technology/building society: Studies in sociotechnical change*, Cambridge, MA: MIT Press, 1992, 205-224.
2. Akrich, M., and Latour, B. A summary of a convenient vocabulary for the semiotics of human and nonhuman assemblies, in W.E. Bijker and J. Law, ed., *Shaping technology/building society: Studies in sociotechnical change*, Cambridge, MA: MIT press, 1992, 259-264.
3. Augustin, L.; Bressler, D.; and Smith, G. Accelerating software development through collaboration, in *Proceedings of The 24th International Conference on Software Engineering* Orlando, FL, 2002559-563.
4. Barley, N. *The innocent anthropologist: Notes from a mud hut*. Long Grove, IL: Waveland Press, 1983.
5. Callon, M. The sociology of an actor-network: The case of the electric vehicle, in M. Callon, J. Law, and A. Rip, ed., *Mapping the dynamics of science and technology*, London: Macmillan, 1986.
6. Callon, M. Some elements of a sociology of translation: Domestication of the scallops and the fishermen of st brieuc bay, in J. Law, ed., *Power, action and belief: A new sociology of knowledge?*, London: Routledge and Kegan Paul, 1986, 196-233.
7. Callon, M., and Latour, B. Unscrewing the big leviathan: How actors macro-structure reality and how sociologists help them to do so, in K. Knorr-Cetina and A.V. Cicourel, ed., *Advances in social theory and methodology: Toward an integration of micro-and macro-sociologies*, London: Routledge and Kegan Paul, 1981, 277-303.
8. Czarniawska, B. *Narrating the organization. Dramas of institutional identity*. Chicago and London: The University of Chicago Press, 1997.
9. Czarniawska, B. *Writing management*. Oxford University Press, 1999.
10. Dinkelacker, J., and Garg, P.K. Corporate Source: Applying open source concepts to a corporate environment, in *Proceedings of Proceedings of the First ICSE International Workshop on Open Source Software Engineering*, Toronto, Canada, 2001.
11. Dinkelacker, J.; Garg, P.K.; Miller, R.; and Nelson, D. Progressive open source, in *Proceedings of Proceedings of the 24th International Conference on Software Engineering (ICSE)*, Orlando, FL., 2002177-184.
12. Fitzgerald, B. The transformation of open source software. *MIS Quarterly*, 30, 3, 2006, 587-598.
13. Foucault, M. *Discipline and punish: The birth of the prison*. Harmondsworth: Penguin, 1977.

14. Granovetter, M.S. The strength of weak ties. *The American Journal of Sociology*, 78, 6, 1973, 1360-1380.
15. Kollock, P. The economies of online cooperation: Gifts and public goods in cyberspace, in M.A. Smith and P. Kollock, ed., *Communities in cyberspace*, London: Routledge, 1999, 220-242.
16. Kollock, P., and Smith, M.A. Communities in cyberspace, in M.A. Smith and P. Kollock, ed., *Communities in cyberspace*, London: Routledge, 1999, 3-28.
17. Latour, B. The powers of association, in John Law, ed., *Power, action and belief. A new sociology of knowledge?*, London: Routledge & Kegan Paul, 1986, 264-280.
18. Latour, B. Technology is society made durable, in J. Law, ed., *A sociology of monsters: Essays on power, technology and domination*, London: Routledge and Kegan Paul, 1991, 103-131.
19. Latour, B. *Aramis, or the love of technology*. Cambridge, MA: Harvard University Press, 1996.
20. Latour, B. *Pandora's hope: Essays on the reality of science studies*. Cambridge, MA: Harvard University Press, 1999.
21. Mauss, M. *The gift: The form and reason for exchange in archaic societies*. New York, NY: W.W. Norton, 1950.
22. Melian, C. *Progressive Open Source: The construction of a development project at Hewlett-Packard*, Dissertation, Stockholm School of Economics, 2007.
23. Nelson, M.L.; Sen, R.; and Subramaniam, C. Understanding open source software: A research classification framework. *Communications of the Association for Information Systems*, 2006, 17, 2006, 2-37.
24. O'Mahony, S., and Ferraro, F. The emergence of governance in an open source community. *Academy of Management Journal*, 50, 5, 2007, 1079-1106.
25. Pettigrew, A.M. Longitudinal field research on change: Theory and practice. *Organization Science*, 1, 3, 1990, 267-292.
26. Raymond, E.S. The cathedral and the bazaar. *FirstMonday*, 3, 3, 1997, Accessed: February 26, 2007, http://www.firstmonday.org/issues/issue3_3/raymond/.
27. Richards, L. *Handling qualitative data: A practical guide*. London: Sage Publications, 2005.
28. Roberts, J.A.; Il-Horn, H.; and Slaughter, S.A. Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects. *Management Science*, 52, 7, 2006, 984-999.
29. Shah, S.K. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 52, 7, 2006, 1000-1014.
30. Smith, M.A. Invisible crowds in cyberspace: Mapping the social structure of the Usenet, in M.A. Smith and P. Kollock, ed., *Communities in cyberspace*, London: Routledge, 1999, 195-219.
31. Steiner, G. *After Babel: Aspects of language and translation*. Third ed., Oxford: Oxford University Press, 1998.
32. Suchman, L.A. *Plans and situated actions: The problem of human-machine communication*. Cambridge: Cambridge University Press, 1987.
33. Tuomi, I. Internet, innovation, and open source: Actors in the network. *FirstMonday*, 6, 1, 2001, Accessed: December 8, 2007, http://www.firstmonday.org/ISSUES/issue6_1/tuomi/index.html.
34. von Hippel, E., and von Krogh, G. Open source software and the 'private-collective' innovation model: Issues for organization science. *Organization Science*, 14, 2, 2003, 209-223.
35. von Krogh, G., and von Hippel, E. The promise of research on open source software. *Management Science*, 52, 7, 2006, 975-983.
36. Weber, S. *The success of open source*. Cambridge, MA: Harvard University Press, 2004.

Mining for Practices in Community Collections: Finds From Simple Wikipedia

Matthijs den Besten¹, Alessandro Rossi², Loris Gaio², Max Loubser³,
and Jean-Michel Dalle⁴

¹ University of Oxford, Oxford e-Research Centre, 7 Keble
Road, OX13QG, Oxford, UK
matthijs.denbesten@oerc.ox.ac.uk,

WWW home page: <http://users.ox.ac.uk/~ierc0002>

² University of Trento, Department of Computer and Management
Science, Via Inama 5, I-38100, Trento, Italy
{arossi,lgaio}@cs.unitn.it,

WWW home page: <http://rock.cs.unitn.it/{arossi,lgaio}>

³ Oxford Internet Institute, 1 St Giles, OX13JS, Oxford, UK
max.loubser@magd.ox.ac.uk,

WWW home page: <http://people.oii.ox.ac.uk/loubser>

⁴ Université Pierre et Marie Curie, c/o Agoranov, 3, rue Castex 75004,
Paris, France

jean-michel.dalle@upmc.fr,

WWW home page: <http://www.upmc.fr>

Abstract. The challenges of commons based peer production are usually associated with the development of complex software projects such as Linux and Apache. But the case of open content production should not be treated as a trivial one. For instance, while the task of maintaining a collection of encyclopedic articles might seem negligible compared to the one of keeping together a software system with its many modules and interdependencies, it still poses quite demanding problems. In this paper, we describe the methods and practices adopted by Simple Wikipedia to keep its articles easy to read. Based on measurements of article readability and similarity, we conclude that while the mechanisms adopted by the community had some effect, in the long run more efforts and new practices might be necessary in order to maintain an acceptable level of readability in the Simple Wikipedia collection.

1 Introduction

Issues pertaining to how communities form, operate, and sustain themselves loom large in any project where voluntary participants gather to create something of value. There are growing concerns regarding which methods and practices within these communities account for their success. This is particularly important if one is interested in replicating successful models to other types of commons-based peer productions. This question is also fueled by quality concerns: if we are to employ a

Please use the following format when citing this chapter:

Besten, M., Rossi, A., Gaio, L., Loubser, M. and Dalle, J.-M., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 105–120.

peer produced collection, we would want to be reassured that what is made available is of sufficient quality and will remain so in the near future. The above considerations hold as much for open source communities developing systems like Linux and Apache as for open content communities creating collaborative repositories like Wikipedia.

Wikipedia is a large-scale collaborative project where voluntary contributors join forces to put together an encyclopaedia that is free for everyone to read and edit. The collaboration has produced, amongst others, an English version with over 2 million articles, a German version with over 600,000 articles and a Simple English version with over 20,000 articles. At the same time, Wikipedia has become extremely visible: for instance, its pages frequently appear high in the result list of search engines [2] and are nowadays commonly referenced by the media. In this paper we take a closer look at the organization of Wikipedia: we speculate on the problems it faces and on the way they are addressed, and, ultimately, we assess its potential for the future: will Wikipedia become an even more established source of information than it already is or will its efforts whither under the weight of vandalism and bureaucracy [7, 14]?

At first sight, Wikipedia is a clear success. On 15 December 2005, *Nature* published a special report comparing the quality of online encyclopaedias. Its findings highlighted that the upstart Wikipedia had a number of errors comparable to Encyclopaedia Britannica [9]. Britannica was compared to Wikipedia for quality by sampling 42 science articles that were sent to experts in the relative field for blind review. While the study was criticized by the Britannica organization for the way articles were selected and compared and clearly suffers from the weakness of a small sample from a large collection, it was influential in establishing that the distributed collaboration on Wikipedia can show performance approximately comparable to traditional methods of encyclopaedia production.¹

In terms of gross production, Wikipedia seems healthy too: content grew exponentially from 2002 to 2005 [18]. However, this is only a rough approximation of performance and does not take into account the quality of the content and as Denning *et al.* [4] note, a proper assessment of the performance and quality of Wiki projects would require attention to many more factors.

With tremendous growth, performance is affected by how well maintenance and organizational work is carried out and how well this non-article related work can be scaled. Reagle [15] examines the way the Wikipedia community develops a neutral point of view in articles and the norms that govern meta-discussions, such as those that take place on article discussion pages. Kittur *et al.* [10] measure directly the impact of these meta-discussions on the performance of the project. They found that the proportion of work on non-article content (such as discussion and fixing

¹ It is worth to note also that while all Wikipedia's inaccuracies raised by the Nature's study has been fixed in a matter of few days (the last being corrected on 25 January, 2006). On the contrary, at least 8 errors (out of the 64 inaccuracies that were not contested by Britannica) still stand nowadays untouched into the current online version of the Encyclopaedia Britannica.

vandalism) is increasing in Wikipedia, showing that more overhead is required as the collection grows and casting doubt over its sustainability.

The increasing demand for maintenance work includes the need to undo the actions of those with malicious intent. In this regard Wikipedia appears to be performing satisfactorily. Even in the early stages of development, it was clear that malicious edits were removed quickly: Ciffolilli [1] argues that it is the nature of the Wiki that makes it easy for friendly contributors to “clean up” vandalism. Later work by Viegas *et al.* [17] showed that “obscene” edits from their sample had a median lifetime of only 1.7 minutes.

Doubts remain over the long-term viability of Wikipedia as a reliable source of information [6, 11]. From the outside, vandalism and well-meaning ignorance are a continuous threat to article quality and from the inside, the dominance of difficult people with lots of time on their hands could easily become a danger of equal if not larger size. In the end, all depends on how well Wikipedia as a community manages to address these threats.

In the paper we take some steps towards studying how organizational practices enable to increase the overall quality of the collection of articles belonging to Simple Wikipedia, a spinoff project of the larger Wikipedia. In order to do so, we have to take into account that communities like Simple Wikipedia might employ methods and practices very different from the ones used in other commons based peer communities. In this sense, our work is very sympathetic with the theory of information goods production developed by Mateos Garcia and Steinmueller [13]. These scholars identify four distinct roles in the production process – that of the author, the editor, the integrator and the conductor – and argue that what distinguishes the “vertical” production of systems like Linux from the “horizontal” production of collections like Wikipedia, is that the cumulative dependencies of components in vertical production make the task of editor and integrator far more valuable and rewarding than the complementary dependencies in horizontal production. What we set out to do for the study reported in this paper was to see if we could identify integration efforts in collections anyway despite the low payoff for volunteers.

In what follows, we report on our findings. We first describe how we selected Simple Wikipedia as a collection, and why it is a good case study to address issues regarding quality and level of integration of the collection. We then lay out what data we have retrieved and how we have mined those data. Finally, we present our findings, which suggest that in the case of Simple Wikipedia that the integration efforts that we were able to identify have some effect but seem insufficient in order to maintain the quality of the collection in the long run.

2 Simple Wikipedia

In order to test our approach, we did a pilot study on Simple Wikipedia. Simple Wikipedia is a spin-off of Wikipedia that was initiated in 2003 because people felt

that many articles in Wikipedia were too hard to read, due to jargon, formality, or for other reasons – especially for children and non-native speakers of English. Simple Wikipedia, was the hope, would be the place where people go to look for an easily readable descriptions of topics. At the same time, contributors to Simple Wikipedia would commit to the ideals of this sub-project or at least adhere to an editorial policy that calls for greater readability when contributing descriptions of topics.

Simple Wikipedia is an ideal candidate to study the performance of commons based peer communities, since it is relatively easy to determine how well it adheres to its goals, zoom in on efforts that are made to address specific problems, and assess the result of these efforts. First of all, with less than 20,000 articles in its collection Simple Wikipedia is a relatively small encyclopedia project. Consequently, no extraordinary computational resources are needed to extract the project archive and analyze its contents. Besides, Simple Wikipedia is a project centered on a very specific goal: readability. What’s more, readability is something that can be measured. So, here we have a project that we can assess on its own terms. Moreover, Simple Wikipedia, as a separate project, was able to experiment and implement with its own editorial policies and managerial policies specifically to help attain its goal of simplicity. Most importantly, Simple Wikipedia has come to rely on the tag “unsimple” – more recently called “complex” – to single out articles, which do not meet its standards and need to be improved.

3 Data

The archive of Simple Wikipedia is available from the web at <http://downloads.wikimedia.org>. Our analysis here is based on the archive of July 2007, which contains the revision history of over 25,000 articles and around 27,000 pages of a different type such as discussion pages and user-pages where regular contributors present themselves. For each edit on an article, the archive lists the IP address or user-name and user-id of the editor, the time of edit, comments made by the editor, and the actual text of resulting from the edit. The text is marked up with tags to identify structural elements like sections and sub-sections and tags of a different type to identify labels – also called templates – that are applied to the text. For instance, an article that is considered to be hard to read will contain the string “`{{unsimple}}`” in the raw text. The status of an editor is slightly harder to determine. We can easily distinguish between editors who are known to the system and editors who contribute anonymously as the latter are identified by their user-id while for the former only the IP address is listed. Bots, scripts that carry out small repetitive edits such as spell checks and interlinking of articles, usually have a user-name that ends with “bot”. In addition, the user-id of these bots is listed as a user belonging to a special group in the auxiliary user-group table, as is the user-id of the users with special rights known as administrators.

3.1 Metrics

In the results that we outline further in this section, we make use of two classes of computational linguistics metrics as a way to characterize changes, made overtime in the Simple Wikipedia articles, resulting from the collaborative and iterative editing process. We are in particular interested in studying two different, albeit interconnected, evolving features of articles: first of all we want to assess how the readability of an article changes over time; subsequently we want to focus on change *tout court* by introducing measures of similarity, accounting for the degree of change/persistence of information in the various revisions of the articles.

The readability of an article is determined by computing the Flesch readability score of the article's text with help of the GNU Style package. This score is a function of the number of syllables per word and the number of words per sentence [8]. More precisely, the formula 'score = 206.835 - 84.6*syllables/words - 1.1015*words/sentences' yields a number that is usually between 0 and 100 and between 60 and 70 for standard English texts. This Flesch reading easy formula, which has been elaborated on the basis of school texts by Flesch in 1948, has been very popular, especially in the US, as a measure of plain English. Its popularity rests on the fact that the formula is easy to compute, yet often accurate. Even word processing applications, such as Microsoft Word, often provide the score as part of their statistics.

The second class of language metrics focuses more explicitly on characterizing the magnitude of persistence and variability of information over time.

The approach proposed here stems from the analysis of text similarities, which is used to compare documents in large text corpora, in order to assess the repetitions of patterns. In this respect, similarity is thus considered a measurable property assessing the degree of relation between two or more information artifacts.

There is a plethora of similarity measures (for an extensive review see [12]) to evaluate this feature; in this study we will take into account two particular vector-based metrics that have been selected for their particular properties.

A measure often used for comparing different documents is represented by the Jaccard coefficient (J), a distance metric defined as follows: given two documents A and B , let a and b the sets of terms occurring in A and B respectively. Define I as the intersection of a and b , and K as their union. Then the Jaccard similarity is the number of elements (cardinality) of I divided by the cardinality of K , thus

$$J = |I| / |K|. \quad (1)$$

Conversely, the Cosine similarity (C) is computed in the following way: let A_s and B_s be sets of terms occurring in A and B , as in the previous measure; define K as the union of A_s and B_s , and let k_i be the i -th element in K . Then the vector terms in A and B are:

$$\begin{aligned} a &= [nA(k_1), nA(k_2), \dots, nA(k_n)]; \\ b &= [nB(k_1), nB(k_2), \dots, nB(k_n)], \end{aligned} \quad (2)$$

where $nA(k_i)$ is the number of occurrences of term k_i in A , and $nB(k_i)$ is the same for B . In this respect, Cosine similarity between two original document sets is defined as

$$C = (a \cdot b) / \|a\| \|b\|, \quad (3)$$

that is the ratio between the scalar product of vectors a and b and their Euclidean norm.

Both measures are widely used for comparisons among documents on broad masses of text corpora, and broadly applied in information retrieval applications and tools [16, 5].

There is a significant difference between the Jaccard coefficient and the Cosine index: both enable to compare two texts, but the former gives more importance to non-overlapped information in the denominator: for this reason, the Jaccard index is much more sensitive to the relative size of documents. Thus, this metric allows to address size differences among documents, while the Cosine metric accounts for the preservation of the same sets of information in different texts. Consequently, when comparing changing versions of the same document in a time domain, the former metric can be used to measure the magnitude of total change, taking into account both the inclusion of new tokens/lemmas and increase/decrease in text size, while the latter signals how much of the former information has been preserved between two versions of a document.

Technically, while the Flesch readability score can be computed directly over the unwikified version of the article (a version cleaned from all wiki tags), further preprocessing of an article is needed before computing the Jaccard and Cosine coefficients: in particular the text needs to be cleaned from stop-words (such as, for instance, propositions and conjunctions) and all remaining words are then truncated into stems.

3.2 Trends

On the basis of the archive, it is possible to reconstruct the history of Simple Wikipedia. For each article in the archive we know when it was first introduced and for every month that Simple Wikipedia existed, we can find the versions of the articles in the archive that existed in that month and we can count their number and properties like their overall readability. Figure 1 in the appendix shows the result of such a reconstruction for Simple Wikipedia from January 2003 until December 2007. The figure shows an upward sloping line indicating the total number of articles in the collection for each month and a downward sloping line indicating the overall readability of the articles in the collection for each month. Note that the continuous growth in the number of articles in Simple Wikipedia is at least partially due to the fact that articles that have been removed from the collection and are not currently available anymore, do not appear in the most recent archive either. Even so, the growth in the number of articles over time is impressive. This rate of growth may be

a factor explaining why the readability indicated by the Flesch readability score shows a gradual decline: as the size of articles in Simple Wikipedia grew, it became more unwieldy and editors faced an ever harder task to maintain the standards of readability, is one interpretation that suggests itself. Looking closely, we can distinguish a phase of substantial decline in readability in the first half of 2004 followed by a more stable phase in the second half of 2004. It was in the second half of 2004 that the practice of tagging articles with the label “unsimple” first appeared. It might be that the slow down in the decline in readability could be attributed to this. With a readability index of over 70, Simple Wikipedia still scores very well given that standard English falls between 60 and 70. Still, the fact that readability continues to decline should be taken seriously into account in the close future.

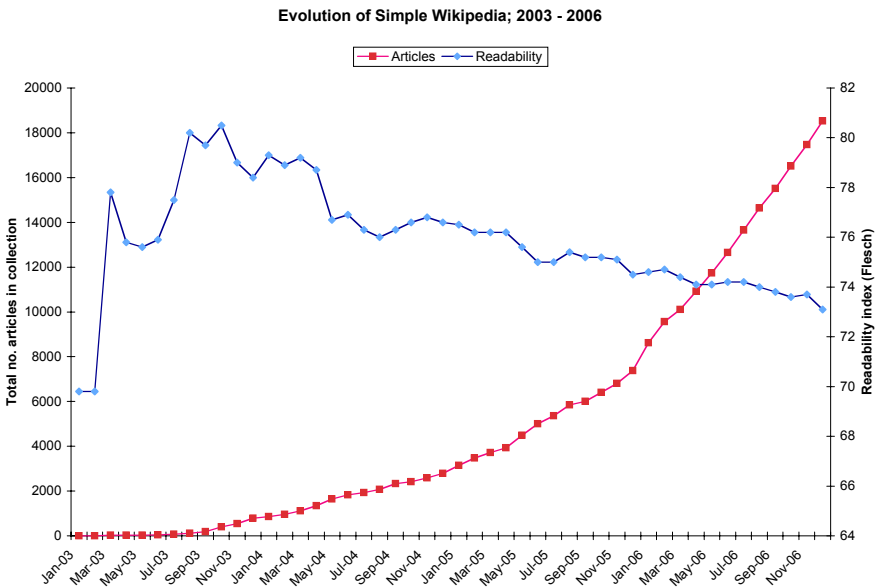


Fig. 1. Performance at Project Level – Simple Wikipedia: total number of articles per month and mean readability index per month.

Figure 2 shows how the similarity indexes change as the articles are revised. Similarity values are averaged for the various articles according to the same revision. Similarity of the n -th revision is computed, respectively, with reference to the first revision of the article (left side) and with respect to the previous revision (right side). Note here and for all subsequent graphical representations of similarity that the metrics are computed over the subset of all the articles which have been labeled as “unsimple” at least once and that the effects of vandal editing have been filtered out in order to improve readability.

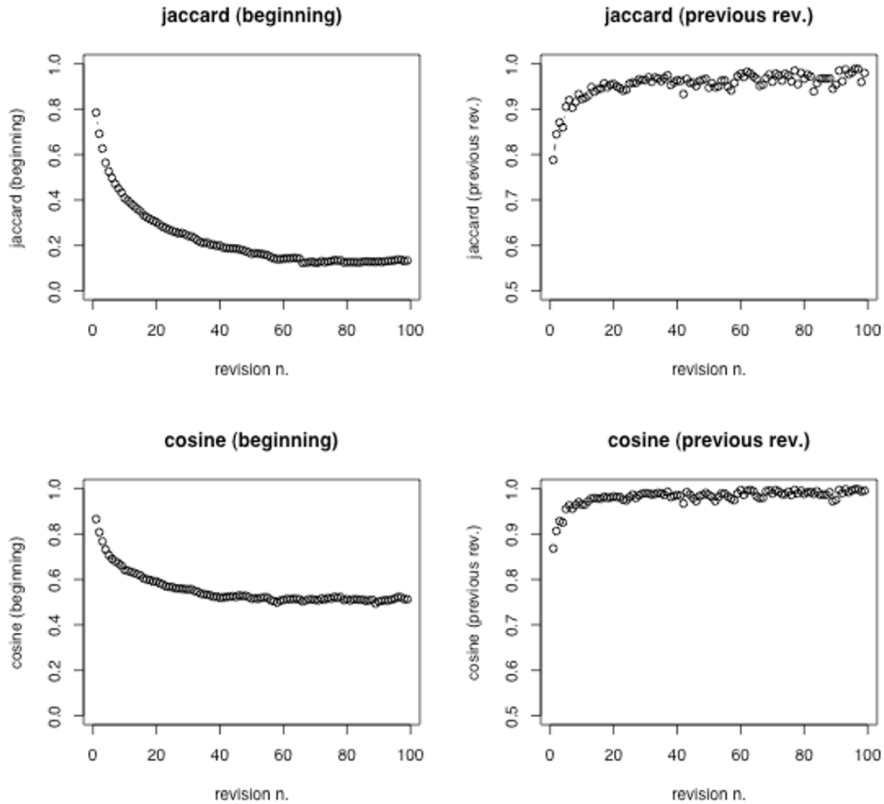


Fig. 2. Evolution of Cosine and Jaccard similarity indexes (vandalisms are filtered out).

The left side of Figure 2 shows that articles on average change considerably with respect to their initial stub and tend to converge to a stable similarity index after circa 40 revisions. Cosine similarity is higher than the Jaccard index, meaning that on average there is a relatively high persistence of initial words/lemmas compared to the high dissimilarity that is introduced overtime by introducing new words/lemmas and by increasing the text size.² The right side of Figure 2 confirms that, as the number of revisions increases, changes become more and more incremental, but since the similarity indexes oscillate at some degree near the lower bound of 1, the revision process still exhibits some, albeit minimal, degree of dissimilarity.³

² This claim takes into account that most articles start as short stub and are enlarged later on, but similar considerations holds also for those articles which are initially copied almost verbatim from en.wikipedia and that are later on simplified and shortened to comply with the Simple Wiki policies.

³ It is maybe worth to note that findings on the left side of the figure are not in contrast with right side ones if one considers that some revisions are circular (as in the frequent case of edit

Figure 3 tries to shed a light on who has been editing articles in Simple Wikipedia from 2004 until 2006. Most dominantly, it shows the growing importance of bots – indicating that automated and semi-automated scripts are relied upon more and more to keep the articles consistent. At the same time, the proportion of edits done by people who do not identify themselves to the system decreases and the proportion of edits do by people who are registered increases relative to them. Whatever causes the gradual decline in readability, it is probably not the hordes of outsiders in the case of Simple Wikipedia.

A final issue is the number of articles that have been identified as “unsimple”. This turns out to be a relatively small number: Less than 200 between 2004 and 2006. We do find however that there is a clear improvement in readability on average following the appearance of the label. Another interesting finding is that the act of tagging seems to be carried out by users with a special administrator status most of the time and hardly by anonymous outsiders (cfr. [3]).

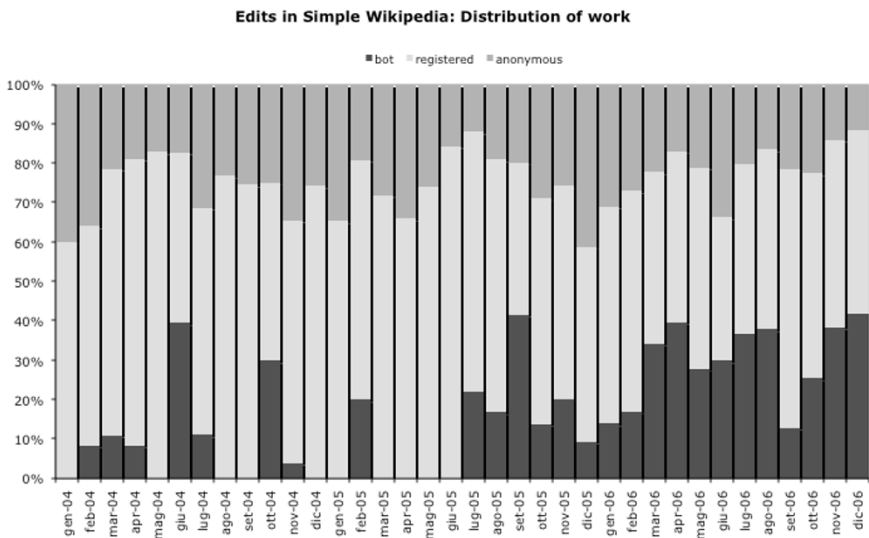


Fig. 3. Variation in editor background over time.

3.3 Triage

Despite the overall trend towards a decline in readability and the limited number of articles that have been labeled with “unsimple”, applying such a label does seem to have the desired effect in several cases. Take for instance the article on propaganda.

warring) and that in the other cases two subsequent revisions, while still presenting some degree of dissimilarity between them, might be relatively equally dissimilar to the initial revision of the article.

Figure 4 depicts the evolution of the readability over time of this page. Readability is based on the Flesch reading easy score and time is given as the number of days relative to the first occurrence of the tag “unsimple” on the page. In addition, the dotted vertical lines indicate the period during which the page was labeled with the tag “unsimple” and the shapes of the points correspond to the different contributors to the page.

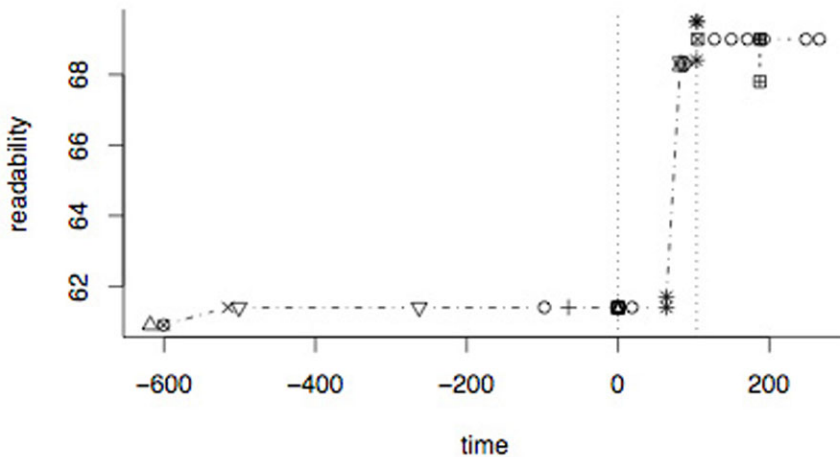


Fig. 4. Treatment of “Propaganda” in Simple Wikipedia: readability index in the “simple” and in the “unsimple” regimes.

The first thing to notice is the dramatic effect that the tag “unsimple” seems to have on the readability of the page. Before the tag first appears, the readability score of the page hovers around the border between a standard and a fairly difficult level of readability. When the tag is removed, 100 days later, the readability has jumped to the other side of the spectrum, now bordering between a standard and a fairly easy level of readability. Looking at the shapes of the points, a history emerges in which someone wrote the initial page to which others then added a bit without changing much in terms of readability. At some point, day 0, a new guy comes along and points out that the page is too difficult to read for an article in the Simple Wikipedia collection. The tag “unsimple” then attracts the attention of someone else who actively starts working to improve the readability of the page. As soon as the problem has been remedied, the tag “unsimple” is removed and regular maintenance is resumed. More specifically the history that this figure represents is as follows (<http://simple.wikipedia.org/w/index.php?title=Propaganda&action=history>): the first version of the article on propaganda appeared on 1 March 2004 and was written by

Randywombat. In the year that followed we see edits by five edits by non-identified users and two by identified users. There are also three edits by bots, but these do nothing more than adding so-called inter-wiki links to similar pages in other languages. In November 2005, the user Heroismic, a high-school junior (according to his/her user-page) who has not edited the page before, comes along and labels the article “unsimple”. In the following months, we see a concerted effort by the users Eptalon, according to his/her user-page a Simple Wiki administrator from Central Europe, and AmanitaMuscaria to improve the page. On 22 February 2006 Eptalon decides that the page has become readable enough and removes the label “unsimple”.

Figure 5 shows the development of readability for few more selected articles. In this figure, the readability of the articles is represented by a ‘o’ while the article has the label “unsimple” and a ‘+’ otherwise. Sometimes readability increases but no one bothers to remove the “unsimple” label. In other cases, the label leads to some improvement in readability initially but contributors fall back on bad habits later. And, sometimes, the label seems to be ignored altogether. The interesting question here is not just how likely it is that this kind of “treatment” by labeling succeeds sometimes, but more why it succeeds in some cases and not in others (for more on this topic see also below in the final section).

Similarly to what we did for readability, we can apply the similarity metrics to selected articles in order to single out different editing processes at work in “simple” vs. “unsimple” regimes. Figure 6 shows the similarity indexes for the article on the year 2001 (“2001”): here for any given revision similarity is computed with respect to the first revision of the ongoing regime. Both Cosine and Jaccard similarity indexes drop abruptly at the beginning; this is largely due to the fact that the article starts as a very short stub on 5/7/04 and it is subsequently substituted at its third revision (on 5/15/04) with a verbatim copy from en.wikipedia. The article remains in the “simple” regime for a total of 37 revisions in the period 5/7/04 – 5/7/06, at the end of which $C=0.32166$ and $J=0.00179$. The unsimple tag is placed by an admin and remains in its place for the subsequent 22 revisions until the article status switches back to “simple” on 12/4/06 for all the remaining available observations (21 revisions up to 6/11/07). If one look closely to the figure, there is a considerable difference in end of regime measures of similarity. At the end of the unsimple regime $C=0.97457$ and $J=0.92732$, while at the end of the second simple regime $C=0.62334$ and $J=0.88344$.

These large differences seem to suggest the existence of rather different patterns of article revision during the “unsimple” regime compared to the “simple” one. In particular, what seems to be particularly counterintuitive is the realizing that the troublesome “unsimple” regime seems to be characterized by a sensibly slower pace of change. While this finding doesn’t hold in principle for all the articles (and calls for more accurate analysis at the single article level), it seems to be confirmed, on average, at the aggregate level.

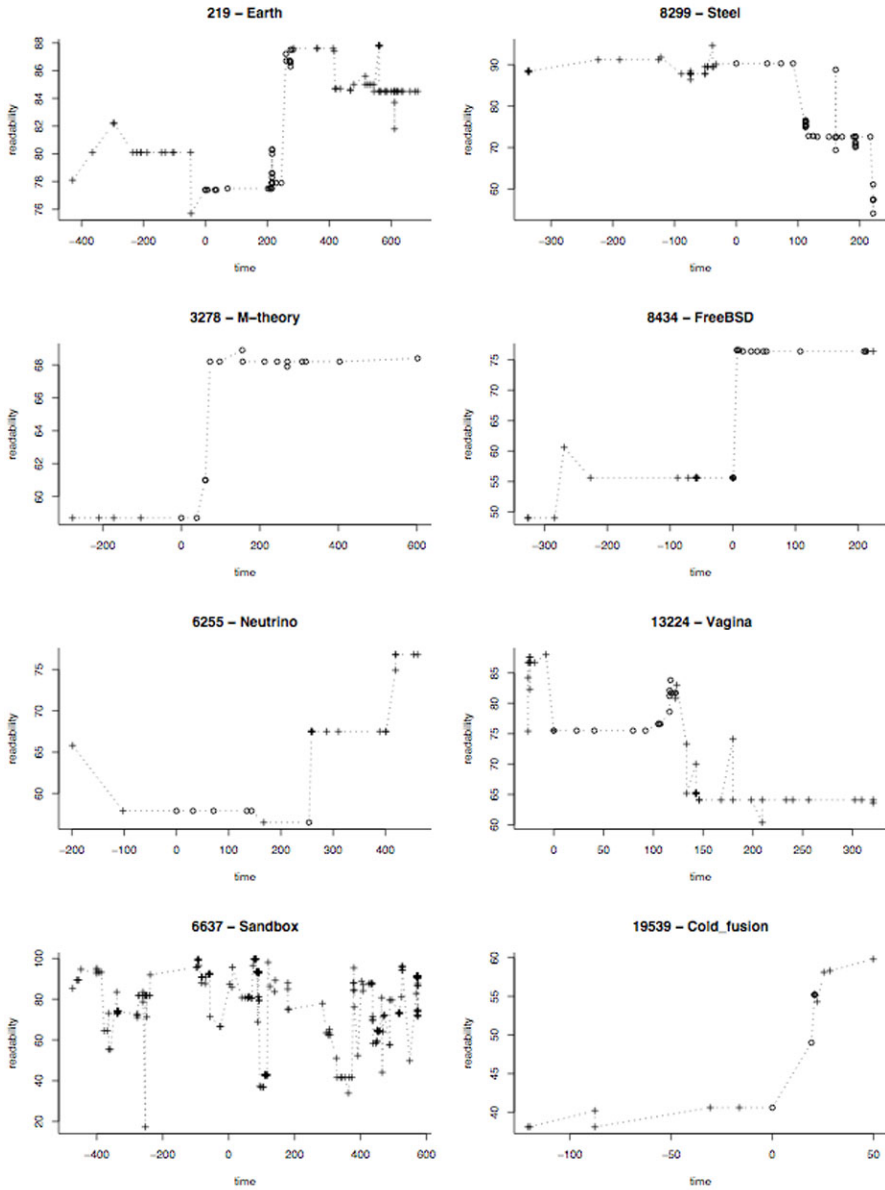


Fig. 5. Development of Readability in Selected Articles.

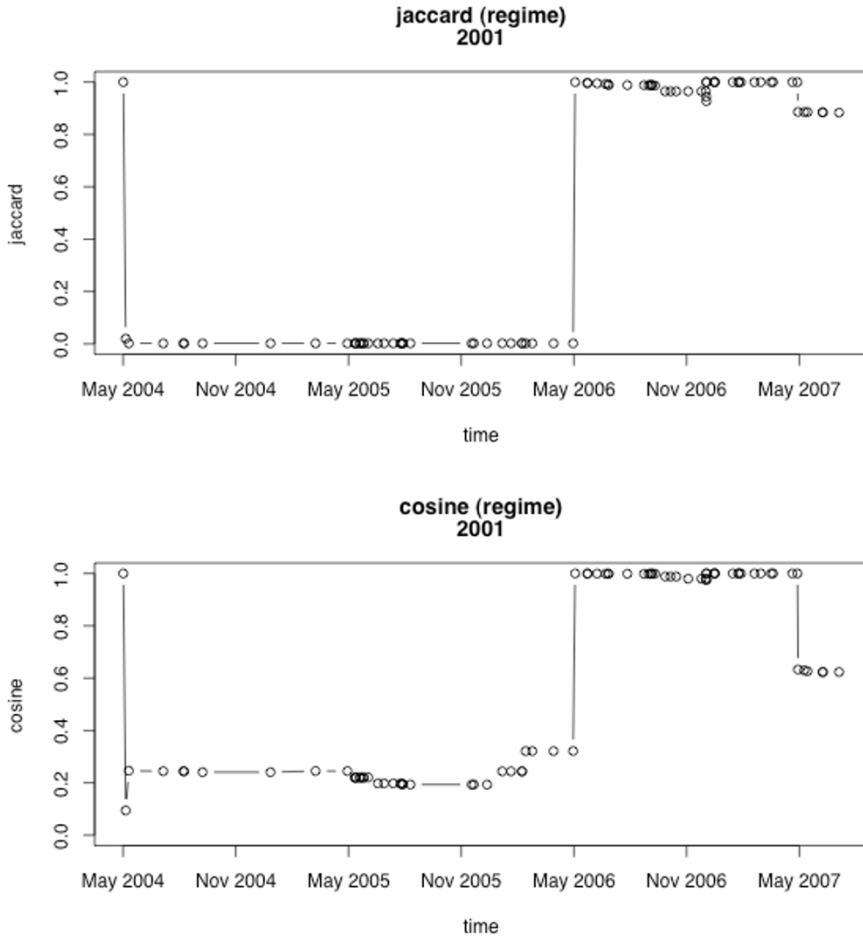


Fig. 6. Treatment of “2001” in Simple Wikipedia: similarity indexes in the simple/unsimple regimes (metrics are computed for any revision with respect to the first revision of the outstanding regime).

Figure 7 shows the distributions of C and J indexes (computed according to the method described for Figure 6) for the first 3 regimes., where regime 1 corresponds to the initial period of editing, regime 2 starts at the first time the article is tagged as “unsimple” and regime 3 starts when the tag is removed (and lasts, eventually, until the “unsimple” tag is placed again). The figure seems clearly to suggest the existence of a two-speed process according to which in the first “simple” regime dissimilarity increase at a very fast pace (mean values: $J=0.549$, $C=0.726$) while revisions in the “unsimple” regime 2 are more incremental (mean values: $J=0.764$, $C=0.9$), and dissimilarity increases again in the subsequent “simple” regime 3 (mean values: $J=0.703$, $C=0.834$). All these differences between two subsequent regimes are

statistically significant (Wilcoxon signed rank test, one sided, $\alpha=5\%$ or lower). Overall, this finding suggests that during non-pathological periods, the editing style seems to favor large revisions making the article progressively more complicated. On the contrary, when the article is tagged as “unsimple”, efforts towards regaining simplicity goes into the direction of more careful and incremental revisions.

We claim that these findings call for further inspection, in particular we find promising to cross readability and similarity analysis looking for a more articulated explanation of the above mentioned differences between “simple” and “unsimple” regimes.

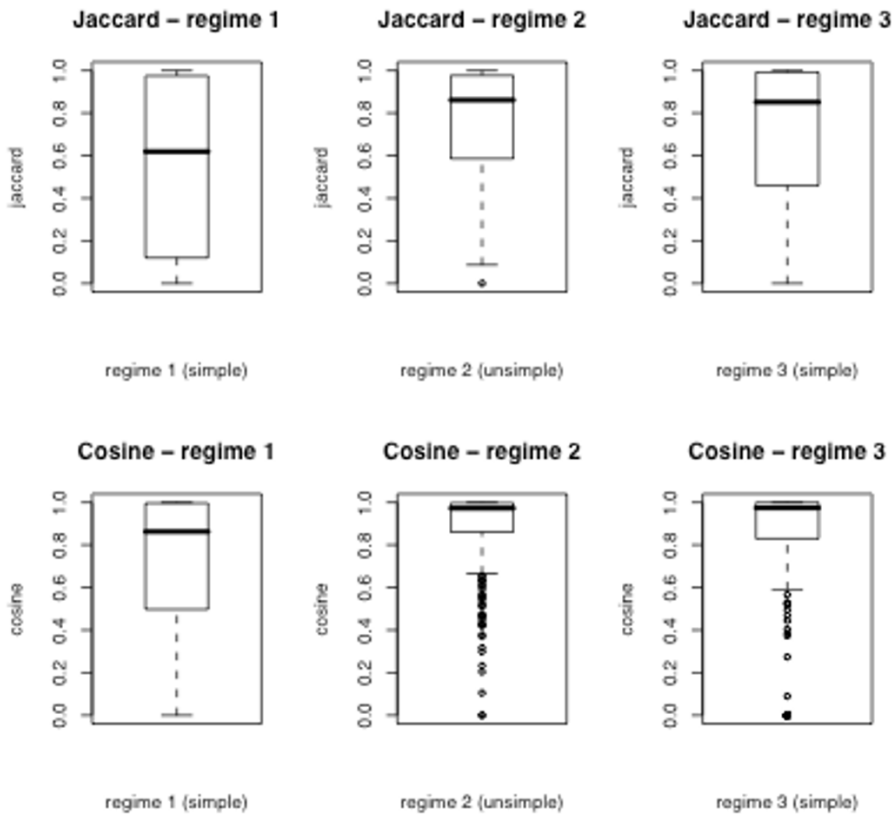


Fig. 7. Box-and-whisker plots of J and C similarity in “unsimple” and “simple” regimes in Simple Wikipedia.

4 Conclusions

The quality and sustainability of Wikipedia is a topic in which scholars of all kinds are getting more and more interested. In this paper, we have provided a review of

recent studies of Wikipedia and we have proposed a new approach to study its performance that takes advantage of the way in which this encyclopedia project is organized. In particular, we propose to look for signs within the Wikipedia archive that specific shortcomings of articles are identified and then assess how well these shortcomings are addressed. We have illustrated our approach with a pilot study of Simple Wikipedia, a spin-off of the main Wikipedia that focuses on articles that are easy to read. We found that while the editors of this collection have a difficult time in keeping articles simple, the label “unsimple” that they employ to single out articles that are particularly problematic still provides a good entry-point in studying how this particular community manages to keep its encyclopaedia readable.

With the Simple Wikipedia study as a “proof of concept”, the time is now ripe to move to other, larger, Wikipedia projects and extend the study to indicators of shortcomings like “controversial” and “stub”. In particular, we find promising to carry out a survival analysis on several variables describing the articles’ state at the point of triage and subsequent edits during the treatment to study the effectiveness of labeling as an organizational practice.

References

1. A. Ciffolilli, Phantom authority, self-selective recruitment and retention of members in virtual communities: The case of Wikipedia, *First Monday*, Vol. 8 (2003) No. 12.
2. J. Čuhalev, Ranking of Wikipedia articles on search engines for searches about its own articles, 2007, <http://www.kiberpipa.org/~gandalf/blog-files/wikistatus/wikistatus.pdf> (last retrieval November 30, 2007).
3. M. den Besten, J.-M. Dalle, Collaborative maintenance of collections – keep it simple: a companion for Simple Wikipedia?, *Industry and Innovation*, Vol. 15 (2008) No.2, pp.169–178.
4. P. Denning, J. Horning, D. Parnas, L. Weinstein, Wikipedia risks, *Communications of the ACM*, Vol. 48 (2005) No. 12, pp. 152–152.
5. I.S. Dhillon, D. Modha, Concept decompositions for large sparse text data using clustering, *Machine Learning*, Vol. 42 (2001) No. 1, pp. 143–175.
6. P. Duguid, Limits of self-organization: Peer production and laws of quality, *First Monday*, Vol. 11 (2006) No. 10.
7. S. Finkelstein, Read me first: Inside, Wikipedia is more like a sweatshop than Santa’s workshop, *The Guardian*, December 6, 2007.
8. R. Flesch, *How to Write Plain English*, Harper and Row, New York, 1979.
9. J. Giles, Internet encyclopaedias go head to head, *Nature*, Vol. 438 (2005) No. 7070, pp. 900–901.

10. A. Kittur, B. Suh, B.A. Pendleton, E.H. Chi, He says, she says: conflict and coordination in Wikipedia, CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, ACM Press, 2007, pp. 453–462.
11. J. Lanier, Digital Maoism: The Hazards of the New Online Collectivism, *Edge*, May 30, 2006.
12. L. Lee, Measures of distributional similarity, Proceedings of the 37th conference on Association for Computational Linguistics, Montreal, Association for Computational Linguistics, Morristown, 1998, pp. 25–32.
13. J. Mateos Garcia, W.E. Steinmueller, Applying the open source development model to knowledge work, INK Open Source Research Working Paper 2, SPRU – Science and Technology Policy Research, University of Sussex, 2003.
14. A. Orlowski, Farewell, Wikipedia? *The Register*, March 6, 2007.
15. J. Reagle, A Case of Mutual Aid: Wikipedia, Politeness, and Perspective Taking, 2004, <http://reagle.org/joseph/2004/agree/wikip-agree.html> (last retrieval May 10, 2007).
16. G. Salton, M. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, New York, 1983.
17. F. Viégas, M. Wattenberg, K. Dave, Studying cooperation and conflict between authors with history flow visualizations, Proceedings of the SIGCHI conference on Human factors in computing systems, 2004, ACM Press, New York, pp. 575–582.
18. J. Voß, Measuring Wikipedia, in: P. Ingwersen, B. Larsen (eds): Proceedings of the 10th International Conference of the International Society for Scientometrics and Informetrics, Karolinska University Press, Stockholm, 2005.

Open to Grok. How do Hackers' Practices Produce Hackers?

Vincenzo D'Andrea¹, Stefano De Paoli, and Maurizio Teli²

¹ Department of Information and Communication Technology

vincenzo.dandrea@dit.unitn.it

² Department of Sociology and Social Research, University of Trento, Italy

{name.surname}@soc.unitn.it

Abstract. How do hackers' practices produce hackers' identities? In this paper we argue that the association between science fiction and software programs is rooted in hackers' practices, defining how hackers' knowledge emerge. The mediation is the one of the Heinlein verb "to grok", part of the Jargon file and of the name of a code browser, OpenGrok, the technology mediating the relationship between OpenSolaris developers and the code base. Starting with a description of the peculiarity of the verb "to grok", and its connection with a non-Cartesian view of knowledge, we discuss how the history of OpenGrok and its use by developers make this knowledge part of hackers' practices and identities, as someone involved in a true, deep understanding of software.

1 Introduction¹

In this article we explore the importance of Science Fiction (SF) in organizing hackers' practices of software development and identities. SF is often part of hackers' everyday life and this claim can be verifiable thanks to several stories and technologies [2],[11].

Nevertheless, research focusing on the role of SF in terms of influencing hacker practices and identities has not been carried out. We maintain that, in hackers' practices, SF has an active role which goes beyond any metaphoric use of it.

Before entering into our contribution we account for two domains of description: identities in FLOSS and the role of SF within organizational studies. The interest of organizational studies for SF is not new, nevertheless in many cases this interest has limited SF to the role of a source of concepts and metaphors, used in the interpretation of concrete cases [1].

For example Srinivas [13] used the concept of Android, inspired by Philip K. Dick's novels. This concept represents the situation of a human being that is no longer entirely human, since he/she finds him or herself caged in a process of commodification. In such sense the contribution provided by Srinivas is that Dick's android becomes a literary resource that can encourage the organizational subjects to withstand the process of "androidization" which subjugate them.

¹ We want to thank Camilla Rossi and the anonymous reviewer.

Please use the following format when citing this chapter:

D'Andrea, V., Paoli, S. and Teli, M., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 121–129.

Even though SF has been found in hackers' story-telling, it has not been studied as a domain for the production of hackers' identities. Trying to understand "who is a hacker?", we found a rational approach to hackerdom, simplified by motivational research ("why do they do that?"), that forgets to consider how hackers' practices produce hackers, focusing only on why they are motivated to participate [12].

Against these uses we claim that the explanatory role of the SF results diminished in the above accounts, because SF is not only a source of interpretative concepts, but it can also be an active part of the organizational processes, contributing to the production of hackers identities. To deepen this claim we focus our analysis on the Martian concept of "Grok", described in the novel "Stranger in a Strange Land" by Robert Heinlein [6]. In order to understand in which ways the concept of Grok participates in software development and use, we studied a software development project known as OpenGrok, paying particular attention to the practices that surround and constitute it.

2 Stranger in a strange land

The novel "Stranger in a Strange Land" narrates the story of Valentine Michael Smith (Mike), a human being raised by Martians on the planet Mars. The novel describes Mike's interactions with the terrestrial culture after its return on Earth after his adolescence. Mike is the child of two members of the first earthling mission on Mars. Because of the death of his parents Mike grew up as a Martian. Mike can then be considered a "man of Mars", in other words he is a human being in his physical structure but a Martian in his mental structure: "He's a man by ancestry, a Martian by environment." [6, p. 20].

The second mission on Mars brought Mike on Earth so that he could be submitted to careful studies. Once arrived on Earth, the condition of Mike is that of a "Stranger in a Strange Land", a Martian man among the humans that observes with Martian eyes the events of which he is protagonist. The novel is focused on the story of Mike, from his initial contact with the human beings, to his understanding of their culture, to the impact of his actions to the human society.

At the same time, in Heinlein's novel emerges that the human beings attempt to understand the knowledge activity of Mike and especially how this activity is mediated by the Martian language. The Martian dictionary articulates around a fundamental verb that is "to grok". This verb, according to the Martian tongue, assumes a multiple meaning: to drink, to understand, to love or "to be one with" a person. To grok means therefore to know a thing or a person in his/her completeness, in the totality of his/her being and at the same time to become part of this thing or person; if we want to express this with a sentence: to grok a thing means "to be one" with that thing.

"'Grok' means to understand so thoroughly that the observer becomes a part of the process being observed [...] and it means as little to us as colour means to a blind

man.” [6, p.266]. Grok separates human beings from Martians, defining different identities and knowledge practices.

2.1 Knowledge and Grok

Why it is interesting to investigate the concept of grok? Firstly, this verb connotes a knowledge that is superior to that perceived or realized by an external observer. The relationship between the observer and the object of investigation is essential here: the knowledge realized by the man of Mars seems to be able to revert the perspective, typical of western modernity, of the separation among who observes and the object that is investigated.

Secondly, the knowledge realized in “grokking” does not seem to have a place in the breast of the Cartesian dualism between subject and object [3]: in grokking activities the observer and observed object become one single thing, which is not part of any of dualism. We can remember that the meaning of grok is to drink and it means to become one with the water that is drunk. Comparing the knowledge realized in grokking - in which a unity is realized between the person and the water - with that of the observer that investigates the water as external object, we can locate the verb to grok in a place at the opposite of the Cartesian perspective.

2.2 Grokking the code

The description provided by Heinlein is a point of departure to show the knowledge dynamics linked with the concept of grok. This task requires an empirical investigation and a discussion of how grok is articulated in practice.

For many aspects, a breakup in the relationship among the observer and an observed object can be found in those practices defined as hacking practices and in the relationship between hackers and calculators. According to the description offered by many [16],[14] this relationship doesn't seem to be instrumental, as that of an external observer towards an object.

Weizenbaum for example describes the differences between “professional programmers”, which think of the calculator as an instrument that is separated from its user, and hackers, that only exist through and for computers [16]. The same rhetoric is presented in Turkle, for whom hackers see computers as “things in themselves”, as opposed to the programmers idea of computers as instruments [14].

Given the consideration above, the verb “to grok” could provide an interesting way to describe hacking practices. Our goal is not to use the verb “to grok” for interpreting hacking practices, on the contrary we would like to observe how this verb is both active part of such practices and how our understanding of such practices would profit from an empirical inquiry.

According to the Jargon File, to grok “Connotes intimate and exhaustive knowledge. When you claim to ‘grok’ some knowledge or technique, you are asserting that you have not merely learned it in a detached instrumental way but that

it has become part of you, part of your identity. For example, to say that you “know” LISP is simply to assert that you can code in it if necessary — but to say you “grok” LISP is to claim that you have deeply entered the world-view and spirit of the language, with the implication that it has transformed your view of programming.”². Thus, a person that groks a certain technology (e.g. LISP) becomes “one with” that technology in a way that this act of grokking has changed radically his/her own personality. Hackers define themselves as involved in grokking software, and grokking involves hackers' identities.

3 OpenGrok: grok in practice

What said so far is not enough to justify our point of view on SF. To understand how the concept of grok participates to software development activities, we studied a development project called OpenGrok, focusing on the practices surrounding and constituting it. OpenGrok is the tool to surf the OpenSolaris operating system code base and, in this way, it participates to the developers' understanding of the OpenSolaris source code. In relation to these practices, the name OpenGrok reminds the association between surfing the code of a FLOSS project and the knowledge translated with the reference to Heinlein's novel. In the following part, we show how this association implies a definition/redefinition of the identities of actors involved, which are allowed by technological mediation to participate to this kind of knowledge; on the other side, we show how the same association between intimate knowledge and this program, is embedded in pre-existing practices and participates to their modification.

In order to study the OpenGrok case we chose to base our theoretical framework on the perspective of the Sociology of associations [7], [8], [9].

We follow the concept of “translation” considered as the interpretation given by technology builders “of their interest and that of the people they enroll” [7, p. 108]. This concept underlines how technological projects are built in attempt to involve more and more people in the projects.

In addition we must consider these practices as “practices of association” where developers build technology mixing together (associating) human and non-human elements.

These acts of enrolling and associating elements is possible thanks to technological mediation. In Latour [8] terms, mediators are entities that “transform, translate, distort, and modify the meaning of the elements that they are supposed to carry” (p.39). Following a mediator helps to understand the definition/redefinition of the elements involved into a translation.

From sociology of associations, we also assume the ethnomethodological perspective. Ethnomethodology [4] is a sociological discipline that aims at understanding the ways people make sense of their world and the ways they produce a shared social order. While other sociologies provide accounts of the social order

² <http://www.catb.org/~esr/jargon/html/G/grok.html>

which competes with that of the people who are members of the society, ethnomethodology aims at describing the practices these people use in their actual descriptions of the social order.

As such, the socio-technical order can be considered as an effect caused by the association between them, and sociology of translation aims at describing the practices that technology builders use in their actual construction of technology.

Finally, we assume that these practices of association are part of an ensemble of pre-existing practices, materials, and textual elements [9], that technologies are not built starting from scratch but they are embedded in pre-existing practices and participate to the modification of such practices.

Methodologically, we observed the discussions taking place in the mailing lists `osol-discuss@opensolaris.org` and `opengrok-discuss@opensolaris.org`, hosted at the OpenSolaris website, we analyzed the blog entries related to OpenGrok, and conducted three email interviews with three OpenGrok developers, the two main developers (CH, TR) and another one (KN). The interview outline has been constructed after the analysis of documents and mailing lists, according to a grounded theory approach [5]. Interviews and observation have been conducted during a two – years long cyberethnography [15] of the OpenSolaris project.

3.1 Toward OpenGrok: a security sentry problems

Initially, we have tried to understand how the association between OpenGrok and to grok emerged, and how it translated the meaning of the Heinlein's verb in practices different from the SF genre. OpenGrok has been developed by CH, a security sentry at Sun Microsystems. In his blog, telling the story of OpenGrok, CH writes: "I keep a watch on reports of newly discovered security holes and then check if any Sun software is affected by them". The relationship between subject (developer) and object (software) involved practices of view and control; these practices involved the participation of another program, `cscope`, allowing a series of searches about the relationship between different functions of a program written in C language. Two elements of `cscope` were considered a limitation in the relationship between programmer and software: first, Solaris is a "Wad of Stuff", with the presence of source files and binary ones; second, some of the programs observed didn't allow access to `cscope` indexes. These boundaries for the reproduction of the sentry status have been the input to create new relationships with the software, new kinds of relationship that brought to OpenGrok.

3.2 Toward OpenGrok: from collecting information to understand them

As shown, OpenGrok (formerly `rob.pl`) has been developed with the aim of enlarging the number of information available for programmers, particularly collecting text strings from binary files; information able to shorten the distance between what a program does (`cscope`) and what a sentry search. Obtaining this kind of information

is not the unique element of “grokking the code”: the practice of control about the presence of security bugs is improved by a search engine able to order the information disposable. CH choice was to use Lucene, “The good thing about Lucene, is that it does not understand document content. You will have to write analyzers for your own content. So you have the control and freedom to interpret different kinds of files the way you want. Lucene does a good job storing your interpretation and searching it.” . The relationship between programmers and the code is mediated by the file interpretation that the same programmer considers relevant. The construction of information useful to reproduce the CH practices is not built as independent from the programmer, but the same programming practices participate to that. The boundary between the object and the knowing subject has been canceled through the mediation of a program which is rooted in and strengthens the unity between subject and object. It seems clear how the association between grok and this program has been sustained by a pre-existing practice, that has defined the need for it and has been redefined by this definition. What has brought to the stabilization of this association in the name “OpenGrok”?

3.3 Toward OpenGrok: relishing the code

It's the same CH, in an email interview, who describes the choice of the name: “We wanted a short, Google unique name [...]. A colleague suggested to use the word 'grok' as it means to fully understand something.” (emphasis added). In this statement, we find another practice, the use of the Google search engine to do web searches, pushing for a short name and an unambiguous recognition of the project. This practice is embedded in the developer field of action, the offices of Sun Microsystems, and the cooperative practices around projects, which are able to associate rob.pl+Lucene to a kind of full understanding, which is summarized by the word “grok”. The analogy with Heinlein story goes on when grok had to be declined to become an unambiguous name; it's still CH telling: “We had initially thought of 'Groktose' to rhyme with Fructose and Glucose. After some discussions with Marketing it became OpenGrok.”. When we asked: “which was the aim of this "sweet" flavour you wanted for your program?”, the answer was “You could through your source code in a tar ball and hope that public will catch it, or you can 'serve' it with OpenGrok and hope people will relish it :-).” Other two elements emerge in connection with OpenGrok and the relationship between programmers and code: first, involving competences and people of the marketing area, the practices to spread OpenSolaris connect the project name with the same operating system, through the use of the prefix “Open”, used also in other Sun's projects; and second, the programmers conceptualization strengthen the idea of physical relationships, like Heinlein's “to drink”, that in this case become “to relish”.

3.4 Beyond Grok: OpenGrok redefines the concept

Something differentiates the SF concept from its practical and discursive actualization: if, according to Heinlein, to grok is a concept humans can't understand, constructing a boundary between Mike and human beings; according to CH, OpenSolaris developer, and the Sun marketing area, OpenGrok, and the possibility to relish the code, can be interpreted as a way to enlarge the number of participants to OpenSolaris, allowing a more direct relationship with the code. They put the full understanding of the system as basic requirement for participation and as a way to overcome boundaries between participants and not-participants, as the references to "public" and "people" make clear. At the same time, the public able to be attracted is not indistinguishable and undetermined, as another OpenGrok developer (KN) underlined in an interview: "Having a name that just says what the product does is too dull, so you try to find a name which both tells something about the product and gives some other associations". The access to OpenGrok and OpenSolaris by the public is mediated by other elements: first, the capacity to interpret the name meaning, as connecting "to grok" to a full understanding; second, other practices can allow the contact between developers and the program, postponing the activity of attribution of meaning to the name. For example, in the KN case, being a Sun employee in Norway, engaged in the development of Apache Derby, as another OpenGrok developer, allowed KN to begin using OpenGrok without knowing the meaning of "to grok": "In fact, I was not familiar with the word "grok" when I started using OpenGrok, so I needed to search for it to find out what it really meant. The exact meaning is still not entirely clear to me, but I think I've understood enough of it to say that it's probably a suitable name. :)". We are able to evaluate another element which makes relevant the study of SF as an active discursive resource: the association between a program and a SF expression constitutes one of the ways through which the same meaning of what hackers are and do is redefined, and put developers in a learning process about "what it really meant".

We can summarize the different ways through which the association "rob.pl + Lucene + grok" participates to hacking practices: a form of full understanding of code is part of the requirement of development practices, like the sentry ones. This knowledge translates in having programs that reduce the distance between programmer and source code, as the choice of Lucene shows; the association of this understanding with the word grok, on one side strengthen the physical relationship between programmer and code, like the suggestion of Grotose shows, on the other side it constructs boundaries around the practices that can produce subjects (hackers) able to participate to this knowledge, practices that do not require to interpret the name OpenGrok. In conclusion, grok associated to a program, translates a meaning rooted in pre-existing programming practices, it gives them new meanings connected with the relationship programmer/code, it also participates to establish how is it possible to access to the practices translated by the same "to grok", and what an hacker is.

4 Conclusions

Our aim with this paper, has been to inquiry in which way SF is an active part of hacking practices, and how its translation as a discursive resource helps defining what an hacker is. We started from the consideration that previous studies looked at SF as a source of interpretative concepts in organization studies, without considering how this can be considered also an active part in organizing processes; and how those studies have considered identities as expressed in motivations, and not as a result of practices.

To empirically strengthen our argument, we have investigated the concept of “grok” as described in Heinlein's novel “Stranger in a Strange Land”, observing how this represents a knowledge practice though which the observer gets blurred and becomes one with the observed object. The shift to an empirical field brought us to consider this kind of knowledge as part of the hacking practices, as described in the Jargon File, a discursive – conceptual resource for the hacker social world. In the following part, we have described an empirical case, the program OpenGrok: its birth; its lexical association with Heinlein's word; and its relationship of anchorage to a translation of hacking practices.

With this empirical analysis, we have shown how “to grok”, taken as an example of the SF imaginary, participates to hacking practices by defining modalities of relationships between the source code of a program and developers, and being itself translated by the participation to the same practices: the hackers' grok allows humans to become Martians, that is to become hackers. Coming back to Heinlein's novel: «the words in English are a mere tautology, empty. In Martian they are a complete set of working instructions.» [6, p.490].

5 References

- [1] A.A. V.V. (1999), Special Issue on Science Fiction, *Organization* 6(4), pp. 579-692.
- [2] Crispin, M. (1978). Software Wars, <http://panda.com/tops-20/software-wars.txt> [13/12/2007]
- [3] Descartes, R. (1637). *Discourse on the Method*, It. Trans. Il discorso sul metodo, 3rd edizione, Paravia.
- [4] Garfinkel, H. (1967). *Studies in Ethnomethodology*. Englewood Cliffs. Prentice-Hall, NJ.
- [5] Glaser, B. G & Strauss, A. L. (1967). *Discovery of Grounded Theory: strategies for qualitative research*, Aldine Transaction, Chicago
- [6] Heinlein, R. (1961). *Stranger in a strange land* – Original uncut version, Penguin, New York, 1991.
- [7] Latour B. (1987). *Science in Action: How to Follow Scientists and Engineers Through Society*, Milton Keynes, Open University Press.
- [8] Latour B.(2005). *Reassembling the Social*, Oxford Univ. Press, Oxford, NY.

- [9] Law J. (2004). *After Method:: Mess in social science research*. London: Routledge.
- [10] Raymond, E. S. (ed.) (2003). The Jargon File. Version 4.4.7, <http://www.catb.org/~esr/jargon/> [13/12/2007]
- [11] Raymond, E. S. (1998). Unix Wars (version 1.1), <http://www.catb.org/~esr/writings/unixwars.html> [13/12/2007]
- [12] Risan, L.. "Hackers produce more than software, they produce hackers," Version 2.1 http://folk.uio.no/lr isan/Linux/Identity_games/ [13/12/2007]
- [13] Srinivas, N. (1999). Managers as Androids: Reading Moral Agency in Philip Dick, in *Organization* 6(4), pp. 609-624.
- [14] Turkle, S. (1984). *The Second Self: Computers and the Human Spirit*, Simon, New York.
- [15] Ward, K. (1999). Cyber-ethnography and the emergence of the virtually new community. *Journal of Information Technology*, 14, 95-105.
- [16] Weizenbaum, J. (1976). *Computer Power and Human Reason*, San Francisco, W.H. Freeman.

Social Dynamics of FLOSS Team Communication Across Channels

Andrea Wiggins, James Howison, and Kevin Crowston
Syracuse University School of Information Studies
Hinds Hall, Syracuse, NY, 13244 USA
(awiggins|jhowison|crowston)@syr.edu
WWW home page: <http://floss.syr.edu>

Abstract. This paper extends prior investigation into the social dynamics of free and open source (FLOSS) teams by examining the methodological questions arising from research using social network analysis on open source projects. We evaluate the validity of data sampling by examining dynamics of communication centralization, which vary across multiple communication channels. We also introduce a method for intensity-based smoothing in dynamic social network analysis.

1 Introduction and Literature Review

Increasing use of social network analysis (SNA) techniques in the research of FLOSS development projects raises concern about the validity of the measured constructs. Centralization is a measure of particular interest for describing the organization of FLOSS teams through the structure of social interactions, as FLOSS team structure has often been claimed by practitioners to exhibit decentralization, while researchers have argued that communication centralization may indicate the kind of strong leadership that enables success in an otherwise decentralized organizational context. However, centralization can vary widely within teams over time and venues [1-5], so an aggregate network structure may provide an overly simplified representation of the interactions in FLOSS teams. In this paper, we present a dynamic approach to assessing project network centralization. To motivate our proposed approach, we first discuss problems with current approaches, namely inappropriate use of measures, non-dynamic analyses, failure to consider intensity of relationships and a focus on a single forum. We then introduce an alternative approach and illustrate its utility in a study of two FLOSS teams.

Research that applies SNA techniques to studies of FLOSS team communication networks (as opposed to team membership networks [3]), typically constructs a social network by using the reply structure of the public threads in a venue as a proxy for direct communication between individuals. This approach defines a link as the interaction between a replier and the immediately previous poster in a threaded

Please use the following format when citing this chapter:

Wiggins, A., Howison, J. and Crowston, K., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 131–142.

discussion. Studies of the information flow characteristics of social networks assume that information flows point-to-point, but depending on the mailing list structure, actual message recipients may include all project participants or all previous posters to the thread, not just the immediately previous poster. The broadcast structure of most of these communication channels therefore restricts the choice of SNA measures that are meaningful, as the potentially public nature of the messages clearly violates the assumptions of information brokerage measures such as betweenness centralization [6], among others. Following [2], we therefore use outdegree centralization [7] as a whole-network measure of inequality of communicative contributions in the network. High values indicate that a few individuals respond to more participants; lower values indicate more equal sharing of communicative work.

Time also presents several challenges for working with these data; all communication networks are sensitive to validity problems from collapsing events over a long period of time. While aggregating events over time is more analytically tractable, it often masks meaningful dynamics and typically fails to retain information about the intensity of relationships [1, 3]. We therefore develop a dynamic analysis of the network. Dynamic analysis requires sampling a time series of snapshots of the networks, based on the time-stamp assigned to the message upon receipt by the message server.

In the projects we studied, periods without any communications were surprisingly common, and this presents a further analytical challenge, as a lack of observations does not necessarily equate to a lack of network structure in the community; people may still have on-going relationships even if they haven't spoken in a particular month. This problem is typically addressed in time-series analysis through smoothing, in which data are divided into overlapping snapshots and sampled in windows (e.g., of 90 days) moving the window forward by a fixed unit (e.g., by 30 days) for each observation [1]. Using a 90-day sliding window, however, means that a single dyad may be reflected in up to three consecutive snapshots; window size is selected to assure that enough observations are present for most forums to generate analysis data for each time period.

A comparison of the effects of smoothing on communication network centralizations is shown in Figure 1; effective smoothing reduces the standard deviation of the network centralizations, but is problematic in that it tends to inflate the mean value. In addition, it tends to "shift" the observations of dynamics forward in time, so that a peak observed in February 2005 with the 30-day window, in Figure 1, only becomes evident in March 2005 with 60-day smoothing, and does not appear until April 2005 when 90-day smoothing is applied.

Intensity of relationships introduces another challenge for SNA in communication networks, for which it has long been known that the strength of ties affects the interactions between individuals [8]. Like most SNA measures, the standard interpretation of outdegree centralization evaluates the centralization of a dichotomous network, those in which ties are either present or absent between any pair of dyads. The measures assume binary relationships because they are designed to evaluate abstract relationships as opposed to the individual communications that

researchers use as a proxy for relationships. In a communication network, a dichotomous representation is a reduction of the available source data.

Most approaches that preserve the information about intensity of interactions in networks employ edge weightings based on unit weighting. Unit weighting increases the weight of each edge by incrementing the edge value by a fixed unit for each

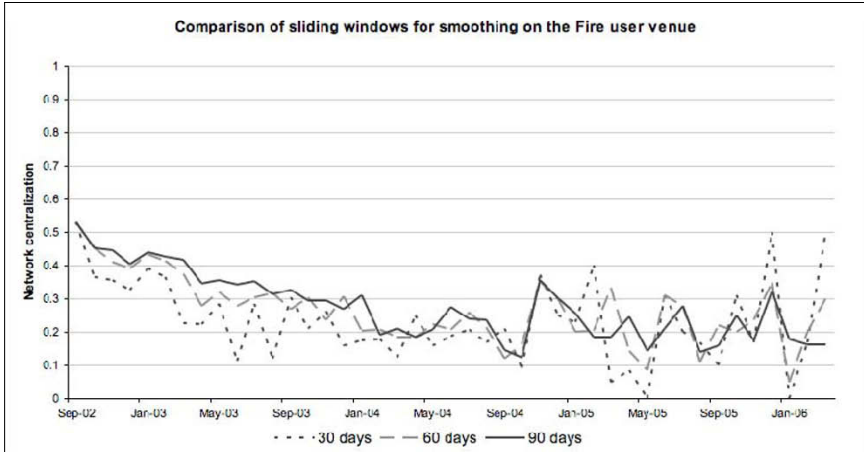


Fig. 1. Comparing the network centralizations observed with different smoothing windows demonstrates the noise reduction from smoothing, such as the period between May and September of 2003. The effect of smoothing also evident for months when no data were available, such as in May 2005, or were very sparse, as in February through April of 2005. For these data, using 90-day periods instead of non-overlapping 30-day periods reduced the standard deviation of the centralizations from 0.14 to 0.12, and increased the mean value from 0.22 to 0.26.

message between a pair in the network sample. Node strength is another option for evaluating centrality with this edge weighting method [9], which indicates the volume of activity in dyadic pairs but assumes a fixed value to each interaction. As few robust measures utilize edge weights, the usual compromise dichotomizes networks based on threshold criteria. This approach allows the analysis of weighted networks using measures that assume dichotomous relationships, but further complicates interpretation by the necessity of selecting threshold criteria, which can be sensitive to such factors as the size of the data sample.

Finally, there are validity considerations inherent in the selection of venues for data sampling. Bug trackers, email lists, and forums have all been used individually for studies of communication networks, but these venues have important differences in function and audience that may affect their interaction dynamics. Many analyses of FLOSS networks analyze only one communication channel to represent the activity of the entire project community. Howison et al. [2] explored the potential for identifying leadership through patterns of contribution to communications, but noted that development contribution is also considered a strong leadership indicator in

FLOSS project teams. Despite theoretical reasons for sampling in particular channels, such as bug trackers or developer email lists, examining the social dynamics of project groups from the perspective of only one communication channel presents an incomplete view of project participation [10]. For example, if we take contribution as a proxy for leadership behavior, we still cannot assume that leadership will be evident in only one of several channels of communication. Participation in bug-fixing, for example, may represent a different form of leadership than participation on a developer or core email list, as seen in analysis of the negotiation processes of bug-fixing [11]. It is therefore reasonable to expect variance in the communication dynamics of user-oriented and developer-oriented venues, and in discussion-oriented versus bug-fixing venues, which poses a potential threat to convergent validity in FLOSS studies that use SNA methods.

2 Data and Methods

In this section of the paper, we present an intensity-based smoothing method to address challenges with dynamic SNA in communication networks and to mitigate the effects of the overlapping window of observations. In our method the recency of a message affects its salience in an ongoing dynamic structure. From this perspective, a more recent interaction has more impact on the current state of the network in each snapshot than an interaction from the very beginning of the time window selected for analysis.

We then examine the validity of venue selection in the following section by comparing the dynamics of communication in multiple venues within projects to determine whether the centralizations of these communication channels show correlated, comparable patterns of change over the course of the project lifespan. Finally we cautiously interpret these measurements for substantive findings in a comparison between two FLOSS projects.

2.1 Sample Selection and Raw Data

Our analysis focuses on the communication patterns in two projects, Fire and Gaim. These projects are similar in that they are both community initiated multi-protocol instant messaging (IM) clients but differ in their ability to sustain project success. Gaim was founded in 1999 as a Linux AOL messenger client and has continued to grow, eventually being ported to Windows and Mac OS X. In early 2006, Gaim changed its project name to Pidgin; our data is selected from the period from the founding of the project in November 1999 until the name change in April 2006. Fire was founded in 2001 on Mac OS X and was initially quite successful, but eventually faced difficulties and made its final release in 2006. Our analysis uses the entire range of Fire's active development lifespan, from 2001 through March 2006.

For each of these projects, communications in the form of email lists, forums, and trackers were obtained from the FLOSSmole [12] and Notre Dame Sourceforge

repositories¹, which collected them from SourceForge. These data were imported into a database (now available through FLOSSmole) to allow automated analysis. We note that FLOSS developers are widely recognized as users of their own products, so we can expect that as developers of IM clients conduct a portion of their communications via IM, and these data are not available for analysis. The observed communication patterns in these projects may therefore differ from the trends that might be seen in projects that are not developing IM clients due to the nature of the product that they develop; analysis to explore this possibility remains for future work. For the Fire project, the communication channels included two trackers, two developer email lists and one user-oriented email list; for Gaim, the channels included four trackers, a user forum, and two developer email lists. For the purposes of comparison, we aggregate individual communications channels into audience-based communication venues because these venues support different types of activities (e.g., discussing programming questions versus user support [10]), making it reasonable to expect that the communication patterns will differ for these groupings of venues.

2.2 Operationalization

To implement intensity-based smoothing for communication dynamics, we developed an original exponential decay function that assigns a weight to each interaction based on its recency and then sums the individual interaction weights to find the edge weight for each dyad. The edge weight decay function shown below is calculated using three dates for each event; the beginning (t_1) and end dates (t_n) for the period, and the date of the event (t_e). The function uses a rate parameter r , determined by the recency of the message event within the total time period t , to scale the value of the message weight w : Let $t = t_n - t_1 + 1$ and $t_{elapsed} = t_e - t_1 + 1$ such that $r = (t - t_{elapsed})/t$. Interaction weights w are given by $w = e^{(-\ln(t)*r)}$ and the interaction weights for each dyad are summed for the edge weight. This assigns the oldest messages in the period a weight that approaches zero, and messages sent on the final day in the period receive a weight of one.

The exponentially decayed weighting is intended to reduce the effects of overlapping windows for data smoothing. To maintain the analytic value of the outdegree centralization measure, the edge weights were subject to a dichotomization threshold so that only edges with values at or above the 0.8 quantile were used to calculate centralization. This threshold selection was made based on sensitivity analysis on a subset of the data and likely affects the effectiveness of the weighting function; an exhaustive comparison of threshold options is a task for future research. Tests comparing the exponentially decayed weighting compared to unit weighting at this threshold saw no effects when applied to a very large data set, such as the Gaim data. Applied to the Fire venues, however, the exponentially decayed weighting

¹ <http://www.nd.edu/~oss/Data/data.html>; access to the Notre Dame repository requires an academic license agreement. FLOSSmole data is publicly available.

showed some variations from the unit weighting, particularly in the less active developer venues, where the correlation between unit and exponentially decayed edge weightings was only 0.86.

The value of exponential edge weighting is best demonstrated on sparse data, for which it provides better smoothing than absolute dichotomization; the venue that showed greatest effect from the use of exponential weighting was also the least active overall. In addition, the exponential weighting better reflects sudden changes in levels of activity from period to period which might otherwise be masked by the use of a unit-weighted smoothing window. Preserving this type of dynamic is beneficial when evaluating the changes to network centralizations. Figure 2 shows the differences between this method and the absolute unit dichotomization method for one email list.

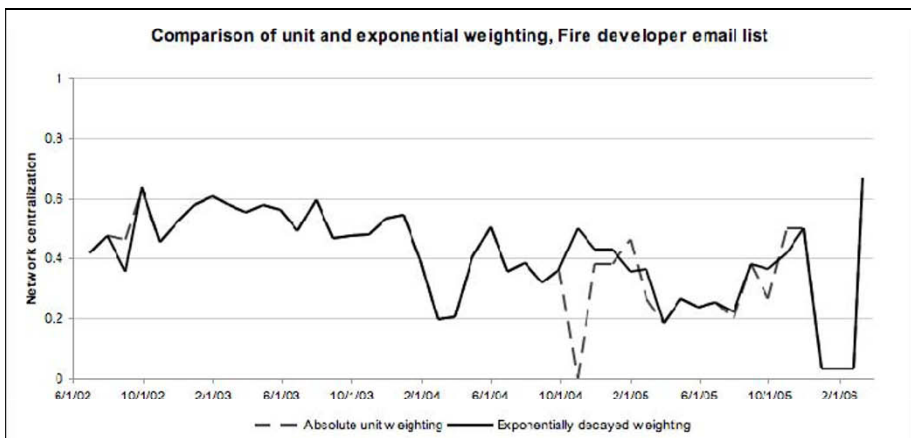


Fig. 2. Comparing network centralizations with absolute unit weighting and exponentially decayed weighting shows no differences for periods of time with higher activity levels, but the exponentially decayed weighting provides better smoothing during periods when there is very little activity.

The dynamic network analysis was performed using a scientific workflow tool, Taverna Workbench, which enabled the development of data analysis workflows that take advantage of modular design and utilize built-in iteration strategies to accomplish a series of data processing tasks over a number of project forum data sets. The workflow used in this analysis was designed to parse mailing list messages into graphs of network centralization over time, depicted in Figure 3². While a brief series of virus messages in one of the venues, identified during content analysis for a separate study, could not be excluded from the current analysis and causes a small effect on one channel, this automated method enabled repeatable analysis of large

² The workflows and XML records of the workflow runs used to produce the analysis are available at <http://ossmole.svn.sourceforge.net/viewvc/ossmole/taverna-workflows/sna/>.

data sets. For example, the Gaim data set included over 41,000 events in the user forum, over 30,000 events in the developer venues, and about 20,000 events in the trackers.

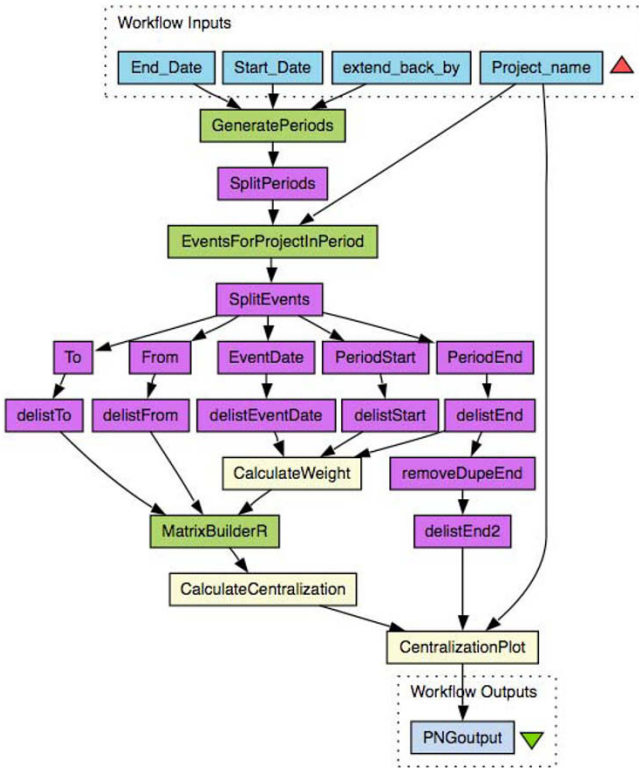


Fig. 3. Taverna Workbench analysis workflow to extract FLOSS message data based on user-provided project name and data sampling time frame for analysis and return an irregular time-series plot of network centralizations. The workflow is explained in further detail in a companion paper for the demonstration of the Taverna tool; see [13] for details.

3 Communication Venues

To examine communication centralization trends across the venues within each project, and between projects, we compare time series analyses for the two projects. Exponentially decayed smoothing was applied to monthly periods with a 90-day overlapping sliding window for the user, developer and tracker venues in both projects. Each project shows different dynamics in each venue; while both projects tend toward greater decentralization in communications over time, they display

varying patterns of interaction. This variation is evident in the correlations between the communication venues within each project.

In three venues for the Fire project (Figure 4) there are comparable mean values for network centralization of trackers and developer email lists (Figure 5). The user email list had a lower average centralization, which reflects the larger and more diverse group of message respondents. The standard deviations of the centralizations are similar for the user and developer venues but higher for trackers due to a spike in centralization values in December of 2005, which affects the values for the following two months due to the sliding window. The sudden change from a very decentralized structure to a highly centralized structure in December of 2005 originates in the feature requests tracker, when one individual closed 279 bugs in a very short period of time. This was most likely in preparation for the end of project development activity, as the final release of Fire followed three months later.

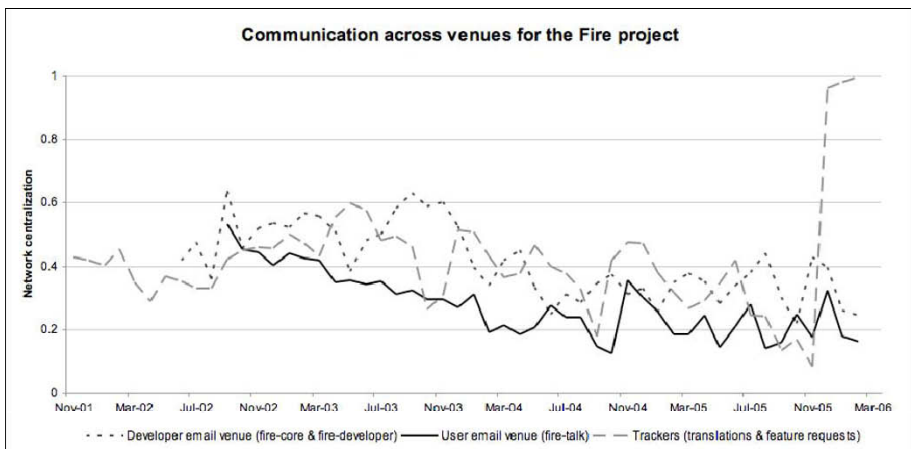


Fig. 4. Communication networks in different venues for the Fire project showed different dynamics over time, although all of the venues show a trend toward increasingly decentralized communications. The tracker shows an interesting exception to this trend just prior to the end of the project activity.

Excluding this period of unusually high centralizations, the mean and standard deviations of the tracker centralizations are comparable to those for the email lists, shown in Figure 5. This may imply some level of regularity across these different venues, and there is an overall downward trend in each venue as communications become increasingly decentralized. Despite these similarities, the three Fire communication venues clearly display different dynamics over time; the correlations in Table 1 show that the developer and user venues are most similar, while the developer venue and the trackers were negatively correlated.

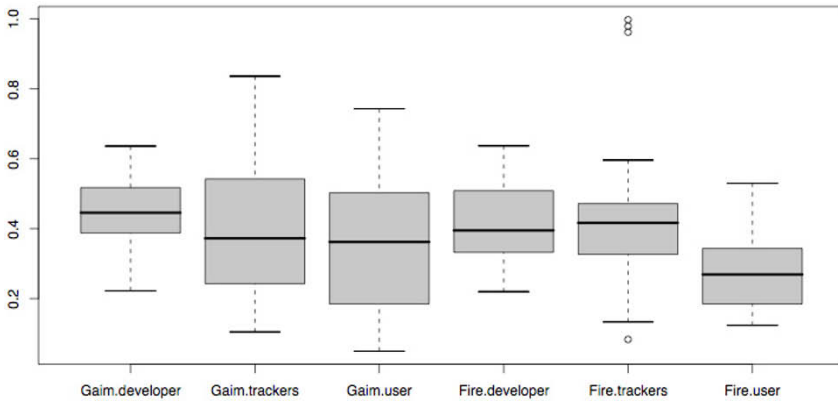


Fig. 5. The distributions for the Gaim and Fire network centralizations show similarity in the summary statistics for the trackers and user venue in Gaim that do not appear in Fire. For each project the user venues have lower centralizations than the developer venues.

Gaim also shows different communication dynamics in different venues (Figure 6); the average centralizations are lowest for the user forum and highest for the developer list, which has a smaller number of participants. The standard deviations of the centralizations for the user forum and the tracker are comparable, while the standard deviation for the developer list was much lower. Visual inspection of the centralization trends reflects a more varied participation dynamic in the user forum and trackers. Periodic spikes in tracker activity appear to indicate project “housecleaning”, much like the phenomenon observed at the end of the Fire project, as there were several periods during which a large number of bugs were closed. These large batches of bug closing were conducted by one (or very few) individuals, generating the observed highly centralized network structures.

Gaim’s user and developer venues both bear greater similarity to the trackers than to one another (Table 1). This result is very different from the correlations for Fire, where the user and developer venues are most alike, and may indicate different uses of these venues by the project participants. One possible explanation for this difference between the projects is that both the user and developer venues were email lists for the Fire project, while the user venue in Gaim was a forum. However, this usage would not explain the very different correlations between activity in developer venues and trackers, or the similarity of the Gaim trackers to the user and developer venues.

Table 1. Correlations of centralizations between communication venues in Fire and Gaim highlight differences in the project communication dynamics.

Project	User-Developer	Developer-Trackers	User-Trackers
Fire	0.62	-0.03	0.21
Gaim	0.17	0.57	0.57

The Gaim venues also show a trend toward decreasing centralization of communications, but the end of the data sample appears to show a more stable range of centralization values. This is confirmed by lower standard deviations for the user and tracker venues, both shifting from approximately 0.18 to 0.9 during the final two years, suggesting that more stable communication patterns have emerged in these venues as the projects matured. At the same time, the developer venue shows little change to standard deviations throughout the project lifespan, which may indicate a different strategy for moderation of development activities over time.

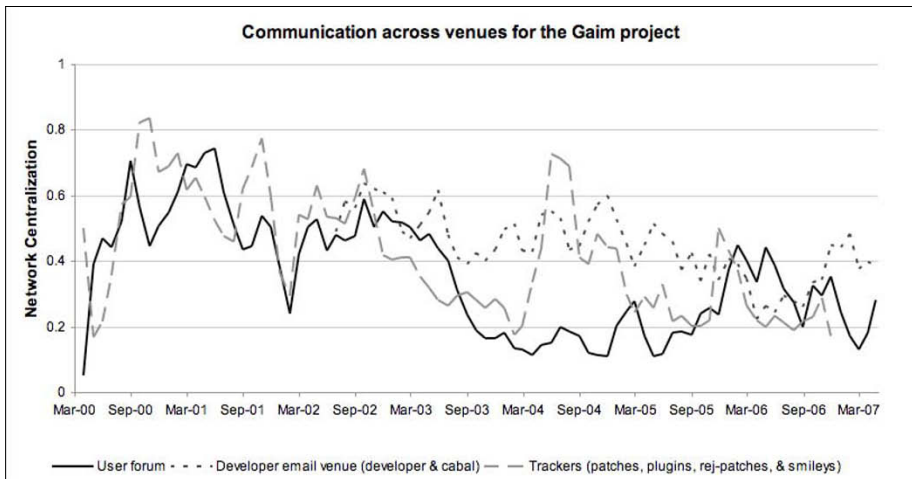


Fig. 6. Communication networks in different venues for the Gaim project also display different dynamics over time; the centralizations for both the user and developer venues were more strongly correlated with the trackers than with one another.

4 Discussion

The use of an intensity-based smoothing assists most with sparse data but the necessity for dichotomization introduces another complication with respect to sensitivity to threshold values, and loses information on the strength of relationships. The selection of smoothing window length may also play a role in the effectiveness of the exponentially decayed weighting, and is another topic for future research.

Analysis of the dynamics across venues shows different levels of correlation between venues for each project, suggesting that these different communication

channels may be proxies for different types of relationships. In both projects, the user venue is more decentralized than the developer venue, reflecting the greater number of participants. Another common feature for all venues in both projects was the overall trend toward decentralization over time, although this could be the result of different influences in each case. In the Fire project, decentralization may be the result of loss of project leadership, while in the successful Gaim project it appears to reflect growth in user participation. Overall, variation in communication dynamics suggests that convergent validity is an important consideration for studies of FLOSS communication networks, and care should be taken in the selection of venues for data sampling, as observations in different venues will generate different results.

In addition, an interesting phenomenon was observed in each of the projects' trackers, where periodic mass bug closings by very few individuals caused sudden, isolated spikes in centralization values. This apparent "housekeeping" behavior, occurring several years into both of these projects, may be a common practice in managing resources for a long-term FLOSS project. We hope to continue the analysis with a larger number of projects to determine whether this phenomenon is common to other projects, which would pose additional challenges to validity for using bug trackers as a data source for analysis of communication dynamics, but which would also point to rhythms in group work, believed to be important to success in distributed environments [14].

5 Conclusion

The dynamic analysis of FLOSS team communications across channels has provided these findings:

- Communication centralization dynamics vary in different venues, suggesting that communication in these venues may be proxies for different kinds of relationships. Researchers should therefore be cautious in using single venues to characterize projects.
- Periodic project management activities in the trackers were evident in both projects, as batch bug closings by a few individuals caused a sudden, temporary shift to a highly centralized network structure. This is both an interesting behavioral phenomenon and a potential confound to analysis of bug trackers.
- All venues in both projects tended toward decentralization over time, a pattern we expect to observe in future analysis of additional projects. Periods in which centralization bucks this trend and rises might be particularly interesting for further study.

This paper also contributes an original method for computing exponentially decayed edge weightings in a dynamic network and makes it available to the research community via a downloadable workflow. Future research could extend this work by examining alternate measures of centrality, and by comparing the individual centralities of developers in multiple channels for each project over time, examining both the dynamics of the individual and aggregated communication channels.

References

1. López-Fernández, L., et al., *Applying Social Network Analysis Techniques to Community-Driven Libre Software Projects*. International Journal of Information Technology and Web Engineering, 2006. **1**(3).
2. Howison, J., K. Inoue, and K. Crowston. *Social dynamics of free and open source team communications*. in *IFIP 2nd International Conference on Open Source Software*. 2006. Lake Como, Italy: Springer.
3. Gao, Y. and G. Madey, *Network Analysis of the SourceForge.net Community*, in *The Third International Conference on Open Source Systems (OSS 2007), IFIP WG 2.13*. 2007: Limerick, Ireland.
4. Braha, D. and Y. Bar-Yam, *From Centrality to Temporary Fame: Dynamic Centrality in Complex Networks*. Complexity, 2006. **12**: p. 59-63.
5. Crowston, K. and J. Howison, *The Social Structure of Free and Open Source Software Development*. First Monday, 2005. **10**(2).
6. Wasserman, S. and K. Faust, *Social network analysis: methods and applications*. Structural analysis in the social sciences ; 8. 1994, Cambridge: Cambridge University Press. xxxi, 825 p.
7. Freeman, L., D. Roeder, and R. Mullholland, *Centrality in Social Networks: li. experimental results*. Social Networks, 1980. **2**: p. 119-141.
8. Granovetter, M.S., *The Strength of Weak Ties*. The American Journal of Sociology, 1973. **78**(6): p. 1360-1380.
9. Valverde, S., et al., *Self-organization patterns in wasp and open source communities*. IEEE Intelligent Systems, 2006. **21**(2): p. 36-40.
10. Sowe, S.K., I. Stamelos, and A. Lefteris, *Identifying knowledge brokers that yield software engineering knowledge*. Information and Software Technology, 2006. **48**(11): p. 1025-1033.
11. Sandusky, R.J. and L. Gasser. *Negotiation and the coordination of information and activity in distributed software problem management*. in *International ACM SIGGROUP Conference on Supporting Group Work*. 2005. Sanibel Island, Florida, USA: ACM.
12. Howison, J., M. Conklin, and K. Crowston, *Flossmole: A collaborative repository for FLOSS research data and analysis*. International Journal of Information Technology and Web Engineering, 2006. **1**(3): p. 17-26.
13. Howison, J., A. Wiggins, and K. Crowston, *eResearch workflows for studying free and open source software development*, in *Proceedings of Fourth International Conference on Open Source Software (IFIP 2.13)*. Milan, Italy. September 2008.
14. Maznevski, M.L. and K.M. Chudoba, *Bridging space over time: Global virtual team dynamics and effectiveness*. Organization Science, 2000. **11**(5): p. 473-492.

Towards a Global Research Infrastructure for Multidisciplinary Study of Free/Open Source Software Development

Les Gasser^{1,2} and Walt Scacchi²

¹Graduate School of Library and Information Science, University of
Illinois at Urbana/Champaign,

²Institute for Software Research

University of California Irvine,

Irvine, CA 92697-3455 USA

+1-949-824-4130, +1-949-824-1715 (fax)

gasser@uiuc.edu, wscacchi@uci.edu

Abstract. The Free/Open Source Software (F/OSS) research community is growing across and within multiple disciplines. This community faces a new and unusual situation. The traditional difficulties of gathering enough empirical data have been replaced by issues of dealing with enormous amounts of freely available public data from many disparate sources (online discussion forums, source code directories, bug reports, OSS Web portals, etc.). Consequently, these data are being discovered, gathered, analyzed, and used to support multidisciplinary research. However at present, no means exist for assembling these data under common access points and frameworks for comparative, longitudinal, and collaborative research across disciplines. Gathering and maintaining large F/OSS data collections reliably and making them usable present several research challenges. For example, current projects usually rely on direct access to, and mining of raw data from groups that generate it, and both of these methods require unique effort for each new corpus, or even for updating existing corpora. In this paper, we identify several needs and critical factors in F/OSS empirical research across disciplines, and suggest recommendations for design of a global research infrastructure for multi-disciplinary research into F/OSS development.

1 Introduction

A significant group of software researchers is beginning to investigate large software projects empirically, using freely available data from F/OSS projects. A body of recent work point out the need for community-wide data collections and research infrastructure to expand the depth and breadth of empirical F/OSS research, and several initial proposals have been made [6, 16, 17, 24, 28]. Most importantly, these data collections and proposed infrastructure are intended to support an active and

Please use the following format when citing this chapter:

Gasser, L. and Scacchi, W., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succì; (Boston: Springer), pp. 143–158.

growing community F/OSS scholars addressing contemporary issues and theoretical foundations in disciplines that include anthropology, economics, informatics (computer-supported cooperative work), information systems, library and information science, management of technology and innovation, law, organization science, policy science, and software engineering. Approximately 200 published studies can be found at the MIT Free/Open Source research community Web site (<http://opensource.mit.edu/>). Furthermore, this research community has researchers based in Asia, Europe, North America, South America, and the South Pacific, thus denoting its international and global membership. Consequently, the research community engaged in empirical studies of F/OSS can be recognized as another member in the growing movement for interdisciplinary software engineering research.

This report attempts to justify and clarify the need for community-wide, sharable research infrastructure and collections of F/OSS data. We review the general case for empirical research on software repositories, articulate some specific current barriers to this empirical research approach, and sketch several community-wide options with the potential to address some of the most critical barriers. First, we review the range of research and research questions that could benefit from a research infrastructure and data collections. Second, we expose critical requirements of such a project. We then suggest a set of components that address these requirements, and put forth several specific recommendations.

2 Objects of Study and Research Questions

As an organizing framework, we identify five main *objects of study*--that is, things whose characteristics researchers are trying to describe and explain--in F/OSS-based empirical software research: *software artifacts and source code*, *software processes*, *development projects and communities*, and *participants' knowledge*. In Table 1 we provide a rough map of some representative characteristics that have been investigated for each of these objects of study, and show some critical factors that researchers have begun linking to these characteristics as explanations. It is important to point out that these objects of study are by no means independent from one another. They should be considered as interdependent elements of F/OSS (e.g., knowledge and processes affect artifacts, communities affect processes, etc.) Also, each of the outcomes shown in Table 1 may play a role as a critical factor in the other categories.

3 Current Research Approaches

We have identified at least four alternative approaches in empirical research on the objects and factors in Table 1 [cf. 50,51]:

- *Very large, population-scale studies* examining common objects selected and extracted from hundreds to tens-of-thousands of F/OSS projects [14, 24, 29, 32, 37] or surveys of comparable numbers of F/OSS developers [21, 25].
- *Large-scale cross-analyses of project and artifact characteristics*, such as code size and code change evolution, development group size, composition and organization, or development processes [7, 18, 23, 39].
- *Medium-scale comparative studies* across multiple kinds of F/OSS projects within different communities or software system types [2, 31, 49, 53].
- *Smaller-scale in-depth case studies* of specific F/OSS practices and processes, for concept/hypothesis development and exposing mechanism details [10, 30, 40, 44, 48, 50, 59].

Table 1: Characteristics of empirical F/OSS studies.

Objects	Success Measures	Critical Driving Factors
Source Code and Artifacts	Quality, downloads, reliability, usability, durability, fit, structure, growth, diversity, localization	Size, complexity, bugs and features, software architecture (structure, substrates, modularity, versions, infrastructure), information architecture, artifact types, and document genres
Processes	Efficiency, ease of improvement, adaptability, effectiveness, complexity, manageability, predictability	Size, distribution, collaboration, knowledge/information management, artifact structure, configuration, agility, innovativeness
Projects	Type, size, duration, number of participants, number of software versions released	Development platforms, tools supporting development and project coordination, software imported from elsewhere, social networks, roles and role migration paths, leadership and core developers, socio-technical vision
Communities	Ease of creation, sustainability, trust, increased social capital, lower rate of participant turnover	Size, economic setting, organizational architecture, behaviors, incentive structures, institutional forms, motivation, participation, core values, common-pool resources, public goods
Knowledge	Creation, codification, use, need, management	Tools, conventions, norms, social structures, technical content, acquisition, representations, reproduction, application

These four alternatives are separated less by fundamental differences in objectives than by technical limitations in existing tools and methods, or by the socio-technical research constraints associated with qualitative ethnographic research methods versus quantitative survey research. For example, qualitative analyses are hard to implement on a large scale, and quantitative methods have to rely on uniform, easily processed data. We believe these distinctions are becoming increasingly blurred as researchers develop and use more sophisticated analysis and modeling tools [e.g., 30, 36, 45, 47], leading to finer gradations in empirical data needs.

4 Available Empirical Data

Increasingly, F/OSS researchers have access to very large quantities and varieties of data, as most of the activity of F/OSS groups is carried on through persistent electronic media whose contents are open and freely available. The variety of data is manifested in several ways.

First, data vary in *content*, with types such as communications (threaded discussions, chats, digests, Web pages, Wikis/Blogs), documentation (user and developer documentation, HOWTO tutorials, FAQs), development data (source code, bug reports, design documents, attributed file directory structures, CVS check-in logs) [48, 51], and programming languages [7].

Second, data originates from different *types of repository* sources [8, 14, 24, 27, 40]. These include shared file systems, communication systems, version control systems, issue tracking systems, content management systems, multi-project FOSS portals (SourceForge.net, Freshmeat.net, Savannah.org, Advogato.org, Tigris.org, etc.), collaborative development or project management environments, FOSS code indexes or link servers (free-soft.org, LinuxLinks.com), search engines (Google.com/code, krugle.org), and others. Each type and instance of such a data repository may differ in the storage data model (relational, object-oriented, hierarchical, network), application data model (data definition schemas), data formats, data type semantics, and conflicts in data model namespaces (due to synonyms and homonyms), modeled, or derived data dependencies. Consequently, data from FOSS repositories is typically heterogeneous and difficult to integrate beyond semantic hypertext linking [41], rather than homogeneous and comparatively easy to integrate.

Third, data can be found from various spatial and temporal *locations*, such as community Web sites, software repositories and indexes, and individual FOSS project Web sites. Data may also be located within secondary sources appearing in research papers or paper collections (e.g., MIT FOSS research paper repository at <http://opensource.mit.edu>), where researchers have published some form of their data set within a publication [40, 52, 60].

Fourth, different *types of data extraction tools and interfaces* (query languages, application program interfaces, Open Data Base Connectors, command shells,

embedded scripting languages, or object request brokers) are needed to select, extract, categorize, and other activities that mine, gather, and prepare data from one or more sources for further analysis [15, 18, 30, 32, 45, 47].

Last, most FOSS project data is available as *artifacts or byproducts* of development, usage, or maintenance activities in FOSS communities. These artifacts/byproducts are a critical part of the FOSS innovation process [61]. However, very little data is directly available in forms specifically intended for research use. This artifact/byproduct origin has several implications for the needs expressed above [16, 49, 51].

5 Addressing Issues with Empirical Data

The main objective of a research infrastructure that collects, aggregates, organizes, and offer data analysis services is to address community-wide resource issues in community-specific way [1, 58]. For F/OSS research, the objective is to improve the collective productivity of software research by lowering the access cost and effort for data that will address the critical questions of software development research. In this section we offer some possible approaches to such an infrastructure, by first briefly describing each “component”, and then considering its benefits and drawbacks.

5.1 Metadata and Meta-models

One of the broadest approaches to common infrastructure is the use of representation standards [1, 59] as meta-data. Such standards would move some issues of cross-source data normalization forward in the process that produces F/OSS projects' information. The use of metadata permits researchers to identify relevant characteristics of specific data collections. Metadata can serve numerous roles in the organization and access of scientific data and documents, including roles in location, identification, security/access control, preservation, and collocation [54]. Standardization of metadata and addition of metadata to F/OSS information repositories, especially at the point of creation, would let the research community identify much more easily the data used in each study, understand and compare data formats, and would also simplify the selection process, by making visible critical selection information [13]. Fortunately, some metadata creation can be automated; unfortunately, representation standards are also an issue for metadata.

Meta-models [38] are ontological schemes that characterize how families of different model sub-types or kinds are interrelated. Meta-models thus provide a critical framework for how to associate and integrate heterogeneous data or metadata sets into a common inter-model substrate. F/OSS tools like Protégé-2000 [43] act as meta-models editors for constructing domain-independent or domain-specific

ontologies, which in turn can produce/output metadata definitions that conceptually unify different data source into common shareable views [30].

5.2 Extracting, Analyzing, Modeling, Visualizing, Simulating, Reenacting, and More with the Available Data

Tools could potentially be developed to address each of the issues reviewed in the previous section. Some such tools already partially exist in a generic form or are developed as needed by research groups. Tools such as web-scrapers that gather data, entity extractors that mine for specific entities like people and dates, or cross-references that link multiple information sources of a single project are commonly developed from scratch in each research effort. These tools are part of the basic toolbox of almost every empirical F/OSS researcher and could easily be provided as such. In fact, several efforts are already underway to produce such tools (e.g.[27, 47]).

Another contribution of a research infrastructure could be to place research data access and manipulation tools upstream, directly within software development tools used by the F/OSS community (e.g., CVS, Subversion, Bugzilla), instead of requiring sometimes-tedious and potentially risky post processing. For example, in most cases, F/OSS tools rely on databases for data storage and manipulation. These databases contain valuable information that is often lost during the translation to a web-visible front-end. (Usually the front-ends rely on web interfaces that display information in a user-friendly fashion but drop important structure in the process). Access to the underlying database can be much more valuable (and in many cases easier) than the current techniques of web-scraping that must recreate such missing relations post-hoc, and may not be successful.

5.3 Integrated Data-to-Literature Environments: Digital libraries and new electronic journal/publishing archives featuring open content and open data.

Putting all the previous components together would lead to a set of normalized, processed and integrated collections of F/OSS data made available to the research community through either federated or centralized mechanisms. These research collections need to be curated and organized as digital libraries that organize and preserve different data sets, meta-data, derived views, analyses, and provenance that span multiple data sets/bases as well as views that span multi-disciplinary models and studies that can be accessed anytime and from anywhere [13, 54]. Furthermore, it should be both possible and desirable to offer subscription and publication services to those who want to be notified when data in the library are changed or updated [3, 12], so that they can re-analyze existing models or derived views.

Finally, an advanced contemporary approach would be an attempt to connect both data sources and research literature in a seamless and interlocking web, so that

research findings can be traced back to sources, and so that basic source data can be linked directly to inferences made from it. Such arrangements provide powerful intrinsic means of discovering connections among research themes and ideas, as they are linked through both citation, through common or related uses of underlying data, and through associations among concepts. Similar efforts are underway in many other sciences (e.g., [1, 58]). Networks of literature and data created in this way, with automated support, can reduce cognitive complexity, establish collocation of concepts and findings, and establish/maintain social organization within and across F/OSS projects. The DSpace repository developed at MIT and Fedora repository [57] are among the leading research candidates that could serve as the storage and archiving facility through which F/OSS data sets, models, views, and analyses can be accessed, published, updated, and redistributed to interested researchers, in an open source, open science manner [cf. 6]. Alternatively, it may be desirable to utilize large, globally accessible repositories, index, and search services as might be provided by a commercial service provider such as Google.

6 Discussion

In our view, the multi-discipline F/OSS research community seeks to establish a *scholarly commons* that provides for communicating, sharing, and building on the ideas, artifacts, tools, and facilities of community participants in an open, globally accessible, and public way [cf. 26, 35]. A shared infrastructure, or in our case, a shared information infrastructure, is a key component and operational facility of such a commons [1, 9, 44]. Such an infrastructure provides a medium for sharing resources of common interest (e.g., F/OSS data sets, domain models, tools for processing data in F/OSS repositories, research pre-prints and publications), common-pool resources (F/OSS portals like SourceForge [55]), and public goods (scientific knowledge, Internet access and connectivity). However, a globally shared information infrastructure supporting F/OSS research may not just emerge spontaneously, though it could emerge in an *ad hoc* manner whose design and operation does not provide for a reasonably equitable distribution of access, costs, or benefits for community participants.

We want to avoid or minimize conditions that make such an infrastructure a venue for conflict (e.g., across disciplines, over data formats, making free riders pay, or rules that limit unconditional access). Consequently, community participants must allocate some attention and effort to address the infrastructure's design and operation. This in turn needs to support and embrace a diverse set of multidisciplinary research interests, but with limited resources. Unsurprisingly, this leads us to a design strategy that is not only iterative, incremental, and continuous, as many F/OSS researchers have agreed [17], but also one that embraces and builds on the practice of F/OSS development practices, processes, artifacts, and tools that are also the subject of our collective research interests. This in part seems inevitable as a

way to address the concomitant need for administratively lightweight governance structures, modest and sustainable financial strategies, and national and international research partnerships among collaborators in different institutions, as well as enabling educational and community growth efforts [9, 26].

7 Recommendations

In accord with the rationales outlined above and the strong sense of the F/OSS community [17, 44], as well as from results developed at the 2008 F/OSS Repositories and Research Infrastructures Workshop (see <http://fossrri.rotterdam.ics.uci.edu>), we recommend that F/OSS researchers begin collective efforts to create sharable infrastructure for collaborative empirical research. This infrastructure should be assembled incrementally, with activity in many of the recommendations defined below.

This paper has provided a sketch of some ideas toward robust and useful infrastructure that can support research within and across the multiple disciplines already investing scholarly effort into the area of F/OSS. The ideas and motivations presented here however need more development, thus collaborative interdisciplinary efforts are encouraged.

Many standards for sharable scientific data exist for other communities, as do many repositories of data conforming to those standards. We should do further research on what other communities have done to organize research data. For example, many collections of social science data are maintained around the world¹. We should use the experiences of these projects as a basis for the F/OSS research infrastructure. The success of these archives in the social science community is also a partial answer to questions of “why bother?” Beyond this, we and others recommend consideration of the following actions that may benefit the global F/OSS research community.

7.1 Instrument Existing Tools and Repositories

We should work with existing F/OSS community development tool projects to design plug-ins to instrument widely used F/OSS tools (such as Bugzilla, CVS/Subversion, etc.) to make the content of those tools available via APIs in standardized formats, administratively controllable by original tool/data owners. Such an effort could also benefit the community of F/OSS developers itself; this sort of instrumentation could help interfacing multiple tools, projects, and communities, and might increase willingness to participate. Further, finding F/OSS projects or multi-project portals that are willing to add support for wide-area event notification

¹ See for example http://www.iue.it/LIB/EResources/E-data/online_archive.shtml for a list of such collections.

services [3] that can publish data set updates to remote research subscribers is real challenge that has been demonstrated to have multiple practical payoffs.

7.2 Continuously Develop Self-Managing F/OSS Research Infrastructure Prototypes

As a proof of concept, we should mock up a complete F/OSS research infrastructure model embodying as many of the desired characteristics as feasible. We should gather data sets, models, analyses, simulations, published studies and more from the many disciplines that are engaged in empirical studies of F/OSS. Such a partial implementation might use, for example, a complete cross section of sharable information from a single project, including chat, news, CVS, bug reporting, and so on. Initial efforts of this kind have produced encouraging results [15, 28, 33]. We and others have already instigated some local efforts in a few of these areas, such as generalized bug report schemas, semi-automated extraction of social processes, preliminary data taxonomies, and automated analysis tools [e.g., 14, 18, 20, 30, 32, 37, 45, 46, 47]. However, there is still a basic need to codify and capture the scientific workflow that FOSS scholars engage in when they analyze FOSS data across many repositories [22].

7.3 Federate FOSS Data Repositories

The European FOSS research community may have more resources (provided by the EC IS&T Programme) already dedicated to development of public FOSS data repositories, data analysis tools, and to the cross-sectional analysis of FOSS data, compared to the U.S. FOSS research community. At a minimum, it would benefit the U.S. FOSS research community to be able to partner with its EU and other international research counterparts to help establish a global FOSS research infrastructure and network of federated FOSS data repositories. Beyond this, it would be of greater research value if the U.S., European, and Asian FOSS research data repositories were collectively federated within a global FOSS research data and community support infrastructure, providing resources to researchers looking to use or host repository data and for FOSS practitioners interested in providing research data.

7.4 Offer Guidance and Incentives for Contributions to a Global Census of FOSS Project Repositories

The diversity and population of FOSS project repositories is unclear and unknown. There is great interest in the research community for a baseline and ongoing census of FOSS project repositories. As FOSS projects choose to collect,

organize, and share the raw data of software development as an activity in their self-interest, then it behooves us within the research community to offer some guidance or incentives for these independent FOSS projects to contribute to such a census. Similarly, we need to articulate what benefits (e.g., socio-economic impacts or intellectual contributions) the research community might offer in return to the FOSS projects that contribute to such a census.

7.5 Identify Research Questions that can Best be Answered through FOSS Data Repositories

The open and public accessibility of raw data from FOSS project repositories and multi-project repositories (e.g., SourceForge.net, FLOSSmole, Google Code [cf. 14, 15, 16, 23]) is providing a new, empirically grounded view of software technology and software development practice—a view that enables comparative, cross-sectional, and ecosystem level studies. This in turn means new kinds of research questions can be posed and new knowledge can be discovered, derived, or created. For example, repository-based studies of successful FOSS projects (of which there are now at least a few thousand such projects) indicate that their software code base, functionality, development artifacts, and developer contributions, and user base can undergo sustained exponential growth, apparently in contradiction to long-standing “laws of software evolution” which traditionally predict sub-linear, inverse square growth rates [cf. 2, 8, 34, 50]. As such, the kind of research questions that follow may ask what model or theory accounts for the super-linear evolution of FOSS systems?

Another example: are there software patterns that constitute a kind of “software genome” that characterize the evolutionary mechanisms of different families of independently developed FOSS systems? Similarly, are the critical software components or functions that lie at the heart of different software families, and does such software represent a critical evolutionary or security vulnerability (e.g., the `glibc` library is commonly bound with FOSS coded in the C programming language)? Also, what development processes best characterize FOSS projects that demonstrate sustained exponential growth of their code and functionality base, as well as the growth of the number of contributors, but with comparable growth/decline of software quality? Last, what can we learn about the evolution of FOSS systems across multiple releases, across multiple platforms, and across different independently developed variants? Exploring any questions like these all require data drawn from multiple FOSS projects or project repositories, and this data is now available. As such, we are on the verge of possible discontinuous advance in our knowledge about software, once again based on empirical studies of FOSS.

7.6 Develop and Share Practices for Curating and Archiving FOSS Project Data

Articulating new knowledge of software products, processes, practices, and organizational forms depends in part on careful and systematic empirical study of FOSS project data. However, this data is not trivial to collect, use, or analyze. As such, there is need to articulate practices for the curation of FOSS project data in forms that increase the likelihood for the data use, reuse, and (re)analysis by people in different disciplines and settings. There is also need to help capture data provenance as well as data annotation and data analysis workflow tools & techniques. Other science disciplines have recognized similar needs, so there is an opportunity for current investments in such areas to be structured to both discipline-specific and cross-discipline research efforts. At present, the FOSS research community has little practice and access to these tools and techniques, and as a result, has little incentive to take on their application or reinvention.

7.7 Identify How Commercial Software Companies can Benefit from Studies Employing FOSS Data Repositories

The commercial software products and service industry in the U.S. is in an awkward strategic position regarding whether or how to take advantage of FOSS, or the results arising from studies of FOSS development data. Software product companies like Microsoft seem hesitant about what to do about FOSS, while software service companies like Google seem to embrace FOSS (as do computer vendors like IBM and SUN). But it appears that all software companies can benefit from empirical studies of FOSS products, processes, practices, and organizational forms that are comparative or cross-sectional, for different competitive reasons.

7.8 Encourage Corporate Sponsors of FOSS Projects to Share their Data with the Research Community

Last, companies like Google, SUN, IBM, and Microsoft Research have established a community of OSS development projects under their corporate sponsorship. These projects are sponsored as a way for these companies to help increase the pool of future software developers who might then transition into the commercial software workforce. These projects also serve to provide a situated, real-world experiment in informal software engineering education, that often takes place outside of the traditional higher education environment. However, “data” from these informal educational experiences is generally not open, nor publicly available, as it is sometimes said to be sensitive, confidential, and proprietary. Thus it is unclear how well these informal experiments work, or whether/how they can be improved both from a corporate perspective as well as from an academic perspective. Perhaps there

is an opportunity to bring together the academic software research and software engineering education community together with the commercial software industry through a government sponsored or coordinated forum so as to articulate how to best advance U.S. socio-economic and scholarly interests for mutual benefit and growth of the software community.

In the end, we believe efforts in these directions identified in these recommendations will pay off in the form of deeper collaborations within and across the empirical software research community, wider awareness of important research issues and means of addressing them, and ultimately in more systematic, grounded knowledge and theory-driven practice in software development.

Acknowledgements: Preparation of this report was supported in part through research grants from the National Science Foundation #0083705, #0205679, #0205724, #0350754, #0534771 and #0749353. No endorsement implied. Collaborators on these projects include Mark Ackerman at University of Michigan, Ann-Arbor, John Noll at Santa Clara University, Margaret Elliott, Chris Jensen, and others at the Institute for Software Research. Megan Conklin, Kevin Crowston, Greg Madey, and others also helped in organizing and contributing recommendations from the *2008 NSF Workshop of Free/Open Source Software Repositories and Research Infrastructures*, (see <http://fossrri.rotterdam.ics.uci.edu>) on which some of the results here are based.

References

1. Bowker G, Baker K, Millerand F. and Ribes D, (2007). Towards Information Infrastructure Studies: Ways of Knowing in a Networked Environment, in J. Hunsinger, M. Allen, and L. Klasrup (eds.), *International Handbook of Internet Research*.
2. Capiluppi A, Morisio M, and Lago, P, (2004). Evolution of Understandability in OSSProjects, *Proc. Eighth European Conf. Software Maintenance and Reengineering (CSMR'04)*.
3. Carzaniga A, Rosenblum D, and Wolf A, (2001). Design and Evaluation of a Wide-Area Event Notification Service, *ACM Trans. Computer Systems*, 19(3), 332-383.
4. Choi SC, Scacchi W, (1990). Extracting and Restructuring the Design of Large Software Systems, *IEEE Software*, 7(1), 66-71, January/February.
5. Conklin M, (2007). Project Entity Matching in FLOSS Repositories, in Feller, J, Fitzgerald, B, Scacchi, W, and Sillitti, A. (Eds.), *Open Source Development, Adoption, and Innovation*, IFIP Vol. 234, Springer, 45-58.
6. David P, Spence M, (2003). *Towards an Institutional Infrastructure for E-Science: The Scope and Challenge*, Oxford Internet Institute Report, September.

7. Delorey D, Knutson C, and Chun S, (2007). Do Programming Languages Affect Productivity? A Case Study using Data from Open Source Projects, *Proc. First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07)*, ACM Press, Minneapolis, MN, May.
8. Deshpande A, Riehle D, (2008). The Total Growth of Open Source, *Proc. Fourth IFIP International Conference on Open Source Systems (OSS2008)*, Milan, IT (to appear, September 2008).
9. Dietz T, Ostrom E, and Stern PC, (2003). The Struggle to Govern the Commons, *Science*, 302, 1907-1912, 12 December.
10. Elliott M, Scacchi W, (2005). Free software development: Cooperation and conflict in a virtual organizational culture. In S Koch, (Ed.), *Free/Open Source Software Development*, Idea Group Publishing, Hershey, PA, 152-173.
11. English R, Schweik, C, (2007). Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of SourceForge.net Projects, *Proc. First Intern. Workshop on Emerging Trends in FLOSS Research and Development*, Minneapolis, MN, May 2007.
12. Eugster PT, Felber PA, Guerraoui R, and Kermarrec A-M, (2003). The Many Faces of Publish/Subscribe, *ACM Computing Surveys*, 35(2), 114-131.
13. Evans, G.E, *Developing library and information center collections*. Libraries Unlimited, Englewood, CO, 4th Edition, 2000.
14. Gao Y, Madey G, (2007) Network Analysis of the SourceForge Community, in Feller, J, Fitzgerald, B, Scacchi, W, and Sillitti, A. (Eds.), *Open Source Development, Adoption, and Innovation*, IFIP Vol. 234, Springer, 187-200.
15. Garg PK, Gschwind T, and Inoue K, (2004). Multi-Project Software Engineering: An Example, *Proc. Intern Workshop on Mining Software Repositories*, Edinburgh, Scotland, May.
16. Gasser L, Ripoché G, and Sandusky R, (2004). Research Infrastructure for Empirical Science of F/OSS, *Proc. Intern. Workshop on Mining Software Repositories*, Edinburgh, Scotland, May.
17. Gasser L, Scacchi W, (2003). *Continuous design of free/open source software: Workshop report and research agenda*, October. <http://www.isr.uci.edu/events/ContinuousDesign/Continuous-Design-OSS-report.pdf>
18. German D, Mockus A, (2003). Automating the Measurement of Open Source Projects. In *Proc. 3rd. Workshop Open Source Software Engineering*, Portland, OR, 63-68, May.
19. GForge, (2008). Gforge Project: A Collaborative Software Development Environment, <http://gforge.org>. Accessed March 2008.
20. Ghosh RA, (2003). Clustering and Dependencies in Free/Open Source Software Development: Methodology and Tools, *First Monday*, 8(4), April.
21. Ghosh RA, and Prakash V, (2000). The Orbiten Free Software Survey, *First Monday*, 5(7).

22. Gil Y, Deelman E, Ellisman M, *et al*, (2007). Examining the Challenges of Scientific Workflows, *Computer*, 40(12), 24-32, December.
23. Gonzalez-Barahona JM, Lopez L, and Robles G, Community Structure of Modules in the Apache Project, Proc. 4th Intern. Workshop on Open Source Software Engineering, 44-48, Edinburgh, Scotland, 2004.
24. Hahsler M. and Koch S, (2005). Discussion of a Large-Scale Open Source Data Collection Methodology, *Proc. 38th Hawaii Intern. Conf. Systems Sciences*, Kailua-Kona, HI, January.
25. Hertel G, Neidner S, and Hermann S, (2003). Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel, *Research Policy*, 32(7), 1159-1177, July.
26. Hess C, Ostrom E, (2004). A Framework for Analyzing Scholarly Communication as a Commons, Presented at the *Workshop on Scholarly Communication as a Commons*, Workshop in Political Theory and Policy Analysis, Indiana University, Bloomington, IN, March 31-April 24.
27. Howison, J, Conklin M, and Crowston K, (2006). FLOSSmole: A collaborative repository for FLOSS research data and analysis. *Intern. J. Information Technology and Web Engineering*, 1(3), 17-26.
28. Huang Y, Xiang X, and Madey G, (2004). A Self Manageable Infrastructure for Supporting Web-based Simulations, *37th Annual Simulation Symposium at the Advanced Simulation Technologies Conference 2004 (ASTC'04)*, Arlington, VA, April.
29. Hunt F. and Johnson P, (2002). On the Pareto Distribution of SourceForge Projects, in C Gacekand B. Eds.), Proc. Open Source Software Development Workshop, 122-129, Newcastle, UK, February.
30. Jensen C, Scacchi W, (2006). Experiences in Discovering, Modeling, and Reenacting Open Source Software Development Processes, in M Li, BE Boehm, and LJ Osterweil (eds.), *Unifying the Software Process Spectrum: Proc. Software Process Workshop*, Beijing, China, May 2005, 442-469, Springer.
31. Jensen C, Scacchi, W, (2007). Role migration and advancement processes in OSSD projects: A comparative case study, in *Proc. 29th Intern. Conf. Soft. Eng.*, ACM, Minneapolis, MN, 364-374.
32. Kawaguchi S, Garg PK. Inoue, K, (2003). On Automatic Categorization of Open Source Software, in Proc. 3rd Workshop OSS Engineering, Portland, OR, 63-68, May.
33. Kim S, Pan K, and Whitehead EJ, (2004). WebDAV: Open Source Collaborative Development Environment, Proc. 4th Intern. Workshop on Open Source Software Engineering, 44-48.
34. Koch S. (2005). Evolution of Open Source Software Systems—A Large-Scale Investigation, in *Proc. 1st Intern. Conf. Open Source Systems (OSS2005)*, Genoa, Italy.
35. Kranich N, (2004). The Role of Research Libraries in Conceptualizing and Fostering Scholarly Commons, Presented at the *Workshop on Scholarly*

- Communication as a Commons, Workshop in Political Theory and Policy Analysis*, Indiana University, Bloomington, IN, March 31-April 2.
36. Lopez-Fernandez L, Robles G, and Gonzalez-Barahona JM, (2004). Applying Social Network Analysis to the Information in CVS Repositories, *Proc. Intern Workshop on Mining Software Repositories*, Edinburgh, May..
 37. Madey G, Freeh V, and Tynan R, (2005). Modeling the F/OSS Community: A Quantitative Investigation, in Koch, S. (ed.), *Free/Open Source Software Development*, 203-221, Idea Group Publishing, Hershey, PA.
 38. Mi P, Scacchi W (1996). A Meta-Model for Formulating Knowledge-Based Models of Software Development, *Decision Support Systems*, 17(4), 313-330.
 39. Mockus A, (2007). Large-Scale Code Reuse in Open Source Software, *Proc. First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07)*, ACM Press, Minneapolis, MN, May.
 40. Mockus A, Fielding RT, and Herbsleb J, (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):1-38, July.
 41. Noll J, Scacchi W, (1991). Integrating Diverse Information Repositories: A Distributed Hypertext Approach, *Computer*, 24(12), 38-45, December.
 42. Noll J, Scacchi W, (1999). Supporting Software Development in Virtual Enterprises, *Journal of Digital Information*, 1(4), February.
 43. Noy NF, Sintek M, Decker S, Crubezy M, Ferguson RW, and Musen MA, (2001). Creating Semantic Web Contents with Protégé-2000, *IEEE Intelligent Systems*, 16(2), 60-71, March/April.
 44. O'Mahony S, (2003). Guarding the Commons: How community managed software projects protect their work, *Research Policy*, 32(7), 1179-1198, July.
 45. Ripoche G, Gasser L, (2003). Scalable automatic extraction of process models for understanding F/OSS bug repair. In *Proc. Intern. Conf. Software & Systems Engineering and their Applications (CSSEA'03)*, December.
 46. Robles G, Gonzalez-Barahona JM, Centeno-Gonzalez J, Matellan-Olivera V, and Rodero-Merino L, (2003). Studying the evolution of libre software projects using publicly available data, in *Proc. 3rd Workshop on OSS Engineering*, Portland, OR, 63-68, May.
 47. Robles G, Gonzalez-Barahona JM, Ghosh R, (2004). GluTheos: Automating the Retrieval and Analysis of Data from Publicly Available Software Repositories, *Proc. Intern Workshop on Mining Software Repositories*, Edinburgh, Scotland, May.
 48. Sandusky R, Gasser L, and Ripoche G, (2004). Bug report networks: Varieties, strategies, and impacts in an OSS development community. *Proc. Intern Workshop on Mining Software Repositories*, Edinburgh, Scotland, May.
 49. Scacchi W, (2002). Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 149(1), 24-39.

50. Scacchi W, (2006). Understanding Free/Open Source Software Evolution, in N.H. Madhavji, J.F. Ramil and D. Perry (Eds.), *Software Evolution and Feedback: Theory and Practice*, John Wiley and Sons Inc, New York, 181-206.
51. Scacchi W, (2007). Free/Open Source Software Development: Recent Research Results and Methods, in M. Zerkowitz (Ed.), *Advances in Computers*, 69, 243-295.
52. Scacchi W, Jensen C, Noll J, and Elliott M, (2006). Multi-Modal Modeling, Analysis and Validation of Open Source Software Development Processes, *Intern. J. Information Technology and Web Engineering*, 1(3), 49-63, 2006.
53. Smith N, Ramil JF, Capiluppi A, (2004). Qualitative Analysis and Simulation of Open Source Software Evolution, *Proc. 5th Intern. Workshop Software Process Simulation and Modeling*, Edinburgh, Scotland, UK, 25-26 May.
54. Smith TR, (1996). The Meta-Information Environment of Digital Libraries. *D-Lib Magazine*, July/August.
55. SourceForge, (2008). <http://www.sourceforge.net> Accessed March 2008.
56. Sowe SK, Angelis L, Stamelos I, and Manopoulos Y, (2007). Using Repository of Repositories (RoRs) to Study the Growth of F/OSS Projects: A Meta-Analysis Research Approach, in Feller, J, Fitzgerald, B, Scacchi, W, and Sillitti, A. (Eds.), *Open Source Development, Adoption, and Innovation*, IFIP Vol. 234, Springer, 147-160.
57. Staples T, Wayland R, and Payette S, (2003). The Fedora Project: An Open-source Digital Object Repository System, *D-Lib Magazine*, April. <http://www.dlib.org/dlib/april03/staples/04staples.html>
58. Star SL, Ruhleder K, (1996). Steps Toward an Ecology of Infrastructure: Design and access for large information spaces. *Information Systems Research*, 7(1):111-134.
59. von Krogh G, Spaeth S, and Lakhani K, (2003). Community, joining, and specialization in open source software innovation: a case study, *Research Policy*, 32(7), 1217-1241, July.
60. Wasserman A, Capra E, (2007). Evaluating Software Engineering Processes in Commercial and Community Open Source Projects, *Proc. First Intern. Workshop on Emerging Trends in FLOSS Research and Development*, Minneapolis, MN, May.
61. West J, Gallagher S, (2006). Patterns of Open Innovation in Open Source Software, Chapter 5 in H. Chesbrough, W. Vanhaverbeke and J. West, (Eds.), *Open Innovation: Researching a New Paradigm*, Oxford, UK, Oxford University Press.

Update Propagation Practices in Highly Reusable Open Source Components

Heikki Orsila¹, Jaco Geldenhuys², Anna Ruokonen³, and Imed Hammouda³

¹ Department of Computer Systems, Tampere University of Technology,
PO Box 553, FI-33101 Tampere, Finland, heikki.orsila@tut.fi

² Department of Computer Science, Stellenbosch University, Private Bag X1,
7602 Matieland, South Africa, jaco@cs.sun.ac.za

³ Department of Software Systems, Tampere University of Technology, PO Box 553,
FI-33101 Tampere, Finland, firstname.lastname@tut.fi

Abstract. In today's business and software arena, more and more companies are adopting open source software. An example of this rising phenomenon is to base software products on highly reusable open source components. In this scenario, the evolution of the software product is coupled with the evolution of the open source component. A common assumption is that component updates are immediately and regularly propagated to the project. This paper investigates this assumption empirically by studying update propagation practices in two popular open source libraries, `zlib` and `FFmpeg`. For each library, we analyze various repository sources with information such as bug reports, revision history, and source code. The results of the case studies suggest that update propagation is subject to several technical and non-technical factors including the way the open source library is used, the extent to which updates are documented, and the degree of community involvement. Based on these findings, we propose a set of recommendations that would allow better follow-up of updates and smoother update propagation.

1 Introduction

Driven by various business and technical motives such as shorter development cycles, lower development costs, improved product quality, and access to source code, more and more software developers and companies are basing their software products on open source components (i.e., libraries, platforms, etc.) [1, 2]. Adopting open source software is sometimes considered a risky business strategy, mainly because of a lack of trust in community-driven software. The main concerns are that many quality attributes such as reliability, security, and safety are hidden properties that have to be carefully checked, and that fixing software defects pertaining to such quality attributes can never be guaranteed. Furthermore, empirical research shows that many advocated hypotheses made about open source software are not always true [3].

Please use the following format when citing this chapter:

Orsila, H., Geldenhuys, J., Ruokonen, A. and Hammouda, I., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 159–170.

A basic practice to overcome part of the challenges sketched above is to regularly update to newer versions of the used open source components, which leads to faster incorporation of community contributions such as bug fixes and new component features. Thus, in the case of highly reusable components, one expects the following basic usage pattern: whenever a new version of a component is released, users of that component immediately switch to the new release. At the other end, one might hypothesize that most practices will eventually deviate from this basic principle due to various influential factors.

Based on the observations above, our research problem can be formulated as analyzing update propagation practices in the case of highly reusable open source components. In particular, we are interested in issues like the frequency of update propagations, the kind of interactions between the components and the projects using them, and the influential factors shaping these interactions. Our goal is ultimately to identify a set of guidelines that would promote better follow-up of updates and smoother update propagation. The good news is that open source projects come with a rich repository of models, source code, resource files, defect reports, change logs, etc., which makes it possible to mine such information.

To investigate our research problem, we focused on two popular software libraries: `FFmpeg` [5] and `zlib` [4]. We carried out our study empirically by analyzing the software repositories of these two libraries for update propagation. The updates concern bug fixes or new features contributed by the community. As expected, mining the information needed was not an easy task as we had to consider various repository sources such as bug reports, revision history and the source code itself. The results of the study show that practices vary from one case to another. In most of the cases, however, we were able to find answers to our research questions and make “educated guesses” for the reasons of the results. Our findings suggest that there are several technical and non-technical factors that have a direct effect on update propagation. These include the way the open source library is being used, the extent of documenting updates, and the degree of involvement in the community.

The rest of the paper is organized as follows. Section 2 presents the background of this work and discusses related studies. Sections 3 and 4 introduce the case studies and empirically explore our research questions. In Section 5 we present a set of recommendations supporting update propagation, and in Section 6 we conclude the paper.

2 Background

The repository sources most relevant to our empirical study are bug reports and revision logs. Most open source projects include an open bug repository, to which users of the software have full access. It is used to report and track bugs and potential enhancements. An open bug repository might potentially increase the number of problems identified in the system and enable more ef-

ficient fixing of problems. Bug reporting, resolving bug reports, and improving bug management have been discussed before [6, 7]. Although many open source software developers interact with the bug repository on a regular basis, there is little data available to characterize their interactions. Similarly, revision logs are useful sources in the sense that they record the evolution of an open source project, but they often come with challenges such as insufficient and unreliable data.

In open source projects, code contribution and bug fixing can be regarded as alternating phases in a continuous, cyclical process. Maintainability has been identified as the core quality issue in open source development [7]: developing an OSS system implies a series of frequent maintenance efforts mainly for debugging existing functionality and adding new features to the system. Maintenance activities can be categorized into four classes: adaptive (e.g., supporting new platforms), corrective (e.g., fixing bugs), perfective (e.g., improving quality attributes), and preventive (e.g., code cleanup and refactoring) [8]. According to this view, open source maintenance is mainly adaptive and corrective. However, in this paper we are more interested in how rapidly the user community reacts to maintenance updates, and in what motivates their reactions.

Reuse of open source software has been the subject of many studies. For example, Capiluppi et al. propose guidelines to identify highly reusable components and to improve the reusability of open source components [9]. Large-scale reuse involving open source repositories has also been studied by Mockus [10]. He identifies widely reused code blocks, typically a component, and common patterns of reuse. In this empirical study the focus is mainly on the immediacy, frequency, usage patterns, and underlying motivation of updates.

In the case of open source components, reuse can be divided (roughly) into the following practices:

- A. Always part of source: the component is incorporated during development time (e.g., the Linux kernel)
- B. Added when released: the component is incorporated during release time (e.g., `xvidcap` project)
- C. User must provide source: the component source code is incorporated by the user when the project is recompiled (e.g., eCos tool chain [11])
- D. User must provide binary: the component binary is provided by the user when the project is linked (e.g., OpenSSH [12])

In special cases, the reuse may even be a combination of two or more of the above (e.g., AbiWord [13]). Reuse of binary distribution makes use of either static or dynamic linking. In static linking the component binary is included in a local, stand-alone copy of the project at compile-time. Dynamic linking means that the component binary is loaded at runtime, and can therefore be updated independently of the project that is using it. In this respect dynamic linking is not relevant to the questions addressed in this paper.

Our research methodology can be summarized in five main steps: we 1) formulate the research questions, 2) select suitable component candidates 3)

extract the relevant data by exploring the component repositories, 4) analyze the data with respect to the questions raised, and 5) make recommendations. The research questions we explore include the following:

- What reuse mechanisms are adopted most often when reusing open source components?
- What kind of update propagation patterns are practiced?
- How fast/often does the user community react to new releases of highly reusable open source components?
- What technical and non-technical criteria influence the community response (e.g., reuse mechanism, product domain, product development phase, etc.)?
- What best practices can be identified to promote better follow-up of updates and smoother update propagation?

As candidate components, we have selected two highly reusable libraries, `zlib` and `FFmpeg`. The former is a lossless compression library which implements a standard coding system [14, 15, 16]; it is used in many file formats and protocols, and in many popular systems such as Linux and Python. The latter is a collection of utilities for processing audio and video files and streams. It includes tools to play and record different media, and a server for distributing media over the internet, for example, for live broadcasts. The library has been incorporated in more than 90 projects.

3 Case Study: `zlib`

3.1 Analysis

The `zlib` source code is included in numerous projects. We looked at three security-related bugs that were found in the `zlib` source code, and analyzed the time it took the bug fix to propagate into 8 projects: `AbiWord`, `BZFlag`, `CVS`, `Linux`, `ppp`, `Python`, `RPM`, and `zlib`.

There are only two core authors for the `zlib` project, but 42 authors have contributed code to the library. Of the 628 documented changes, 89% come from the top 5 out of 42 contributors. This information comes from the credits in the library's `ChangeLog` file. We studied the latest `zlib`, version 1.2.3 released on 2005-07-18. The `ChangeLog` entries are dated 1995-04-11 to 2005-07-18, a period of approximately 10 years.

We investigated fixes for the following bugs:

1. *A double free bug* reported on 2002-03-11
2. *A DoS/crash bug* reported on 2004-08-25
3. *A buffer overrun/DoS/crash bug* reported on 2005-06-30

For each of the projects that use `zlib`, the bug status was classified as follows:

- **Does not apply:** The bug doesn't have an effect on the project, because the vulnerable code never existed inside the project (e.g., Linux kernel)
- **Known:** The time (in days) to fix a bug is known from version history (e.g., CVS)
- **Not fixed:** The bug is still not fixed (e.g., AbiWord for Windows)
- **Unknown:** Status of the fix is unknown due to unavailability of version history (e.g., Python)

The results are shown in Table 1. The mean and median times for fixing a bug, computed over all the projects in the table, are 97 and 19 days, respectively.

Table 1. Number of days to fix 3 different `zlib` bugs

Project	Bug 1	Bug 2	Bug 3
AbiWord	1	Not fixed	Not fixed
BZFlag	Does not apply	Does not apply	583
CVS	1	63	87
Linux	8	Does not apply	Does not apply
ppp	21	Does not apply	Does not apply
Python	Unknown	Unknown	90
RPM	432	25	16
<code>zlib</code>	0	15	11
Min	0	15	11
Mean	77	34	157
Median	5	25	87
Max	432	63	583

3.2 Discussion

We noticed two issues from the `zlib` results:

Bug Fix Delay Varies Significantly The time to fix bugs varies a lot from project to project, which means that the median and mean times to fix bugs are far apart. In one case (`AbiWord`) the project is still vulnerable for bugs that were discovered years ago.

We did not find any project, apart from one operating system distributor, with an explicit system for checking for updates in other projects. Such an automatic mechanism is a necessity for scalable code reuse. Possible reasons for this lapse may be weak virtual organization, lack of explicit task lists, and lack of command hierarchy. Another possible reason that stabilized products fail to incorporate `zlib` updates is that their maintainers do not have the resources for testing a new `zlib` version and/or backporting necessary fixes.

Investigated projects seem to fall into three update propagation patterns: random updates, negligence, and systematic. We address these issues further in the guidelines in Section 5.

Microsoft Windows Programs Are Biased Towards Binary And Source Duplication Table 2 shows each project and its reuse category. Almost all GNU/Linux projects use `zlib` as a dynamic library, which can be updated through a common packet manager for all applications. Unfortunately the Microsoft Windows operating system lacks a common packet manager for non-Microsoft products. This suggests that there could be a general bias in Microsoft Windows to use source duplication (category A and B) instead of dynamic libraries (category D), because it is so much harder to update those systems.

Table 2. Projects and their reuse categories

Project	Reuse categories
AbiWord	A, D
BZFlag	A, D
CVS	A, D
Linux	A
ppp	A
Python	A
RPM	A, D
zlib	A

For example, `AbiWord` and `Python 1.6-2.4` can be run on both GNU/Linux and Microsoft Windows, but the Windows versions are more vulnerable to bugs because the GNU/Linux versions use a dynamic library. `Python 2.5` and later are in category A, and therefore they are vulnerable on both GNU/Linux and Microsoft Windows. Bug 3 is one manifestation of this problem.

4 Case Study: FFmpeg

4.1 Analysis

FFmpeg has a core library called `libavcodec` that contains encoders and decoders for a wide range of multimedia formats. The FFmpeg project web page lists some 90 other projects that incorporate parts or all of FFmpeg. We focused our attention on `libavcodec` and, in particular, the library interface specification in the header file `avcodec.h`.

Material related to this case study is available at [17].

Figure 1 shows the development of `avcodec.h`. Each dot represents one change, and its color identifies the responsible user. During the period 2001-07 to 2007-06, 38 different users made a total of 617 changes and the file grew from 177 (5.1 kbytes) to 2940 (90 kbytes) lines of code. Only the most active users are listed in the figure.

As a first approximation we studied the revision history of `avcodec.h` in the following projects:

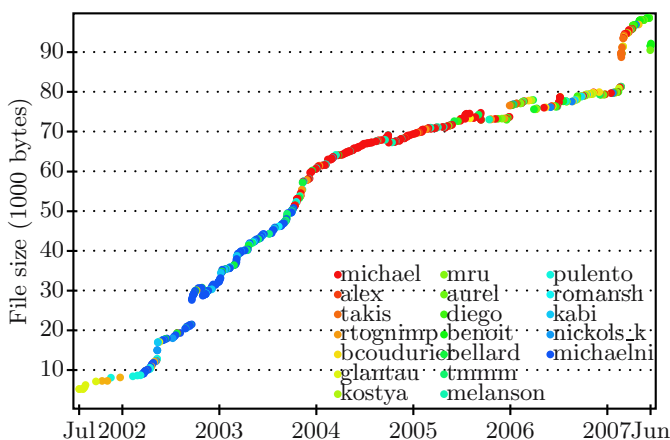


Fig. 1. The evolution of `avcodec.h`

- `avidemux` is a video editing suite
- `avifile` is a multimedia player for Linux systems
- `ffdshow` is a media decoder and encoder for Microsoft Windows systems
- `gststreamer` is a server for streaming audio/video over the internet
- `mythtv` is a software-based personal video recorder for Linux and Mac OS X
- `xbmc` is a multimedia player for Microsoft’s Xbox

Unfortunately, it soon became clear that the revision history by itself is not sufficient. Log entries such as “`libavcodec resync`” do not identify the exact revision of `libavcodec` that was used for the update. Even when such information is given, there is no guarantee that the comment is accurate, and in several cases it proved not to be. To overcome this problem, different revisions of `avcodec.h` were downloaded and compared one by one to find matching versions. Together with the information in the revision logs, this produced the kind of information shown in Figure 2. The upper and lower horizontal lines represent the `avifile` and `FFmpeg` projects, respectively. Arrows indicate updates from the latter to the former on the dates shown above and below the project lines, and the small vertical lines on the `FFmpeg` line indicate different revisions of `avcodec.h`.

Table 3. Summary of update data

Project	Period	Nr. of updates	Delay (days)		
			Min	Max	Ave
<code>avidemux</code>	2004-01–2007-01	10	1.8	26.8	5.7
<code>avifile</code>	2002-05–2007-05	163	<hour	14.6	2.1
<code>gststreamer</code>	2004-03–2006-09	9	1.1	18.0	5.2
<code>mythtv</code>	2002-08–2007-06	82	<hour	60.6	3.7
<code>xbmc</code>	2004-04–2007-04	7	2.9	118.7	29.8

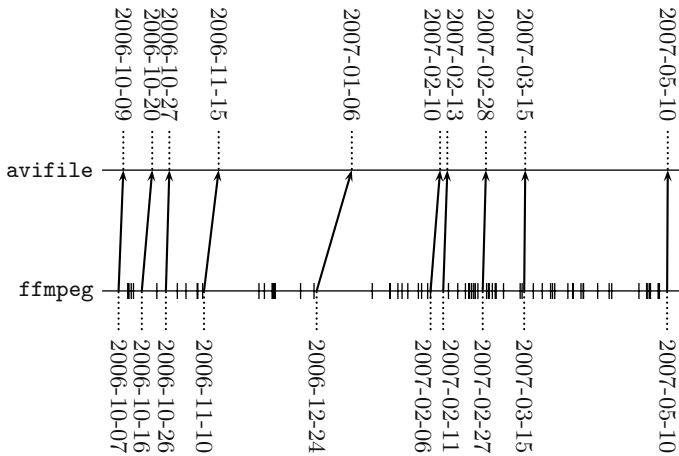


Fig. 2. The 10 most recent updates (from 2006-10-09 to 2007-05-10) of `avcodec.h` in `avifile`

The results are summarized in Table 3. In the three rightmost columns, *Delay* refers to the number of elapsed days between the production of a revision and its use in an update, in other words, how “fresh” it is. In many cases, the update delay is less than a week. `avifile` and `mythtv` were clearly updated much more frequently than the others, even when the longer time frames are taken into account. Still, on at least one occasion the `mythtv` project was not updated for about two months. One fact not shown in this table is that these two projects are updated less and less frequently over time, possibly because `libavcodec` has reached a level of stability where fewer bugs are reported.

4.2 Discussion

We noticed some issues from the `FFmpeg` results:

Shared Interests, Features and Developers New features in the `avifile` project were introduced first in child projects and later introduced into the parent project. This is the result of common developers and interests between the projects. For example, one active developer is a member of both the `FFmpeg` and `avifile` projects.

Feature propagation to and from the parent project supports the theory of OSS development model. If new features are only added in the central project, it would cast serious doubt on the OSS model, and distributed models in general. Bugs are often fixed through inter-project co-operation by users and developers directly communicating with each other. We argue that code reuse in some OSS projects comes close to sharing developers, not just sharing features (code).

Update Propagation Entails Significant Effort In the case of `mythtv` the mismatches were more numerous and more substantial. The requirements of this

software include specialized features such as closed captioning and support for multilingual soundtracks, which are not provided by the `libavcodec` library. In at least one case, a feature was first introduced in the `mythtv` project and only later in `FFmpeg`, but, as far as we can tell, the later implementation was not derived from the earlier one. In all events, `mythtv` is one example of an unforeseen pattern: code reuse takes place, but the code undergoes non-trivial modifications within the new setting, requiring significant effort in update propagation. This happens either because the new setting is significantly different from the original, or because there are new feature requirements.

On Update Propagation Patterns All the projects mentioned above include a complete copy of the `libavcodec` code and fall into category A. Sometimes, as in the case of `mythtv`, this is unavoidable if the code needs modification before use.

One example of a project that belongs to category B is `xvidcap`, a screen capture program for recording user activity, to create video tutorials and other material. Whenever this project is released, the developers include the latest version of `libavcodec`. This may appear safer, since the library is not as tightly integrated in the source code. However, category A reuse allows users to download a more recent, albeit less stable, development version of the software. In theory, users could themselves replace the copy of `libavcodec` in `xvidcap` with the latest release in order incorporate the latest bug fixes, but it is not realistic that this would occur widely in practice.

In the case of `ffdshow`, we failed to trace updates accurately, because the project modifies `avcodec.h` too dramatically, but we observed that at least some bugs were reported back to the `FFmpeg` project.

Comparison of `zlib` and `FFmpeg` Bug fix delays in the `zlib` project were relatively long compared to those in `FFmpeg`. The main difference in these projects is that functionality of `zlib` is fixed; it just implements a specific functionality. In contrast, new features are continuously added to `FFmpeg`. This suggests that that shared interests in developing the same features, as well as shared developers, can decrease the update propagation delay.

5 Guidelines for Managing Bug Fixes

Based on our experience of tracking bug fixes we argue for some guidelines for managing bug fixing inside and between open source projects.

5.1 Avoid Source and Binary Code Duplication

Use dynamic libraries instead of static libraries or source code inclusion. Source code inclusion means that reused code has to be constantly maintained and monitored. In general, dynamic libraries avoid redundant information in the system. This is by far the best practice for code reuse, because it also allows other parties, mostly operating system distributors, to help you.

5.2 Document Important Changes in Version Control History

Maintain a special **SECURITY** file that lists the specific corrective maintenance operations in version control history that fix a security issue. This helps operating system maintainers and other interested parties to track important updates. Also, backporting the security fixes to older and stable versions is easier when the specific commit is known. This is important for production systems that operate for several years. This approach is not limited to security fixes. It can also be used to document other bugs, properties, or interesting factors.

5.3 Tag Important Changes in Version Control History

A special tag (e.g., “[**SECURITY**]”) should be added to each commit message that fixes a security issue in the version control system. This makes searching for security fixes easier.

5.4 Maintain a Global Notification System for Changes

Fast updates between projects is important if new features are needed or security matters. Achieving this demands easy updates. We propose that each project create a global notification system for important changes (e.g., a mailing list), that alerts interested parties of specific fixes and features. The system should not flood interested parties about small changes, only the important ones. Also, notifications should be archived so that users can access older changes.

5.5 Facilitate Follow-up of Component Updates

To address the issue of weak command hierarchy, we propose that each project creates a list of reused software components, annotated with a timestamp and an unique identifier (such as the version number or a commit identifier) about the last update.

Also, if possible, a responsible person should be assigned to participate actively in the project community of the reused component. This will decrease update propagation time and may help to promote a project’s interests. This happens when companies contribute to projects like Linux, gcc and Samba.

5.6 Write a Procedure for the Update Process

Virtual organization and distributed development means that any developer should be able to replace another developer — at least in theory — but unfortunately experience and knowledge is not easily transferred. A detailed set of guidelines to raise awareness about maintenance operations should help new developers to be more productive. Such guidelines could, for example, spell out how to cross-check for important updates, such as security fixes.

Guidelines should cover issues relating to project maintenance and releases, including managing updates (notification of changes to and from other projects), managing new releases, and the preferred communication style between developers (IRC channels, mailing lists, etc.).

6 Conclusions

Many software companies and software developers are adopting open source components. These components often undergo constant maintenance actions. This paper studied the causes of maintenance in open source components and what controls the user community reaction to maintenance updates. We studied update propagation delay. In particular, we analyzed the effect of the following factors on update propagation delay: reuse category, documentation of changes, and the update process itself.

To find answers we explored updates and bug fix delays in the `zlib` and `FFmpeg` software repositories. We found that update propagation delay varies significantly among projects. Based on this information, we formulated the following guidelines for reusing open source components:

1. Avoid source and binary code duplication.
2. Document important changes in version control history.
3. Tag important changes in version control history.
4. Maintain a global notification system for changes.
5. Facilitate follow-up of component updates.
6. Write a procedure for update process.

Since we have only studied update propagation in the context of two libraries, we cannot claim that the results are generalizable. For further investigation, more case studies should be considered. In our case studies we used custom-built scripts and analysis by hand; a full record of our experiments is available at the website [17]. Although the details are specific to the libraries we looked at, the approach is applicable to other cases, and scalable to larger projects. If the guidelines we have suggested are followed, our approach would be even easier and faster. In order to validate the relevance of the proposed guidelines, a questionnaire to the open source community could be planned and carried out.

References

1. Bolado, M., Castillo, J., Posadas, H., Sanchez, P., Villar, E., Sanchez, C., Blasco, P., Fouren, H.: Using open source cores in real applications. In: DCIS 2003. (2003), 683–688
2. Madanmoha, T., Deapos, R.: Open source reuse in commercial firms. *Comm. ACM* (2004), 62–69

3. Paulson, J.W., Succi, G., Eberlein, A.: An empirical study of open-source and closed-source software products. *IEEE Trans. on Softw. Eng.* (2004), 246–256
4. zlib web site: <http://zlib.net>
5. FFmpeg web site: <http://ffmpeg.mplayerhq.hu>
6. Anvik, J., Hiew, L., Murphy, G.C.: Coping with an open bug repository. In: *eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, ACM Press (2005), 35–39
7. Samoladas, I., Stamelos, I., Angelis, L., Oikonomou, A.: Open source software development should strive for even greater code maintainability. *Comm. ACM* (2004), 83–87
8. Lientz, B.P., Swanson, E.B.: *Software Maintenance Management*. Addison-Wesley (1980)
9. Capiluppi, A., Boldyreff, C.: Coupling patterns in the effective reuse of open source software. In *Proceedings of the 1st International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07)*, IEEE Computer Society (2007)
10. Mockus, A.: Large-scale code reuse in open source software. In *Proceedings of the 1st International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07)*, IEEE Computer Society (2007)
11. eCos tool chain: <http://ecos.sourceware.org/build-toolchain.html>
12. OpenSSH web site: <http://www.openssh.com>
13. AbiWord web site: <http://www.abisource.com>
14. Internet Society RFC 1950: ZLIB Compressed Data Format Specification version 3.3, <http://tools.ietf.org/html/rfc1950>
15. Internet Society RFC 1951: DEFLATE Compressed Data Format Specification version 1.3 <http://tools.ietf.org/html/rfc1951>
16. Internet Society RFC 1952: GZIP File Format Specification version 4.3 <http://tools.ietf.org/html/rfc1952>
17. <http://www.iki.fi/shd/publications/oss2008/>, and a full dump of the web site: <http://www.iki.fi/shd/publications/oss2008.tar.gz>

Using Social Network Analysis Techniques to Study Collaboration between a FLOSS Community and a Company

Juan Martinez-Romo¹, Gregorio Robles², Jesus M. Gonzalez-Barahona²,
and Miguel Ortuño-Perez²

¹ Dpto. Lenguajes y Sistemas Informaticos, E.T.S.I. Informatica (UNED)
juaner@lsi.uned.es

² Grupo de Sistemas y Comunicaciones, Universidad Rey Juan Carlos
{grex,jgb,mortuno}@gsyc.esct.urjc.es

Abstract. Because of the sheer volume of information available in FLOSS repositories, simple analysis have to face the problems of filtering the relevant information. Hence, it is essential to apply methodologies that highlight that information for a given aspect of the project. In this paper, some techniques from the social sciences have been used on data from version control systems to extract information about the development process of FLOSS projects with the aim of highlighting several processes that occur in FLOSS projects and that are difficult to obtain by other means. In particular, the collaboration between the FLOSS community and a company has been studied by selecting two projects as case studies. The results highlight aspects such as efficiency in the development process, release management and leadership turnover.

1 Introduction

Software projects are usually the collective work of many developers. In most cases, and especially in the case of large projects, those developers are formally organised in a well defined (usually hierarchical) structure, with clear guidelines about how to interact with each other, and the procedures and channels to use. Each team of developers is assigned to certain modules of the system, and only in rare cases they work outside their *territory*. However, this is usually not the case in FLOSS projects, where only loose (if any) formal structures can be recognised. On the contrary, FLOSS developers usually have access to any part of the software, and even in the case of large projects they can *move* more or less freely from one module to another with only some restrictions imposed by the common uses in the project, and the rules on which developers themselves have agreed. A large amount of spontaneous interaction structures arise, evolve and disappear without the intervention of a central control, yielding complex networks.

Among complex networks, social network analysis (SNA) appear as a method for analysing the structure and interactions of people and groups of

Please use the following format when citing this chapter:

Martinez-Romo, J., Robles, G., Gonzalez-Barahona, M. and Ortuño-Perez, M., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 171–186.

people within extensive organisations. A vast amount of scientific works show applications of these techniques in fields that go from the social sciences to physics and computer networks [20, 10].

The goal of this paper is to apply classical social network concepts to data extracted from FLOSS projects, in particular to FLOSS projects that show a tight collaboration between a company and the FLOSS community. As, even in the presence of a company, FLOSS projects have a loose management style and are based heavily on third-party contributions, social network analyses may provide with some insight into the underlying processes that are responsible for the development. In this way, we have an ad-hoc network of interpersonal dependencies that come from the interactions between the nodes that integrate it. Therefore, this work is mainly descriptive and has the purpose of showing how the application of techniques of social networks can be useful in scopes beyond the original ones.

The rest of this paper is organised as follows. The next section describes related research on this topic. Next, the methodology designed to extract data from source code management systems and a series of technical aspects on which we have based our study is presented. Afterwards, a brief historical review of the main events within the projects will be given and then the results are shown and discussed. A final section with some conclusions and hints about further research closes the paper.

2 Related Research

Much attention in the area of research on FLOSS projects has been focused on the organisational structure of the projects [9, 17, 15], but little attention to the dynamics of the group of developers. A noteworthy contribution in this sense, although not directly addressing the evolution of developer communities, is the *onion model* [5], which shows how developers and users are positioned in communities. This model differentiates among core developers (those who have a high involvement in the project), codevelopers (with specific but frequent contributions), active users (contributing only occasionally) and passive users [18, 7, 13].

The onion model provides only a static picture of a project, lacking the time dimension that is required for studying processes, among these the ones of joining and leaving a project. A more theoretical identification and description of the roles, including also some dynamics [26], has helped to fill the gap. According to this refinement, a core developer is supposed to go through all the outlying roles, starting as a user, until she eventually reaches the core team. Some research has tried to measure how long this process takes for a volunteer participant (i.e. somebody who is not hired by a company to work on the project), obtaining that the time that passes from the first e-mails to the mailing lists (considered as the first participation) to the first commit (considered as the developer becoming part of the core group) is in the mean of

around 30 months [11]. An alternative approach has been proposed [12] after studying and modelling the processes of role migration for some FLOSS communities, focusing on end-users who become developers. This has led to the identification of various paths for the joining process, concluding that the organisational structure of the studied projects is highly dynamic in comparison to traditional software development organisations. The attraction of human resources to FLOSS projects has also been analysed [6, 24], with different models proposed about how developers enter new projects.

With respect to abandonment, the number of Debian developers leaving the project has been studied [22], and how this affects its evolution (i.e. what happens to software packages that become unmaintained because of the abandonment). The authors propose a half-life parameter, defined as the time required for a certain group of contributors to fall to half of its initial population, which is of 7.5 years for the Debian project.

3 Methodology

3.1 Construction of the Developers Network

The methodology used in this study is based on retrieving data about the activity of developers from the source code management repository of the project [19, 27]. In the case of FLOSS projects, this usually means either a CVS or a Subversion repository, which is mined using CVSAly [23]. This tool retrieves the information about every commit to the repository, and inserts it into a database where it can be conveniently analysed. This information includes, for each commit (modification in a file in the repository): the date, the username of the developer (committer), and the number of lines involved.

Once the information has been stored in the database, we proceed to construct the developer network. Each vertex represents a particular developer and two vertices will be linked by an edge when both they have contributed, at least, one common software artifact¹. Edges may be weighted by means of a *weight of the relationship*, defined for instance, as the total number of commits performed by both developers on artifacts to which both have contributed. Therefore, a relation between two developers (vertices) exist when both have worked in the same artifact, and a link will exist between them (edge) whether both have made, at least, a *commit* in the same artifact.

3.2 Indexes

When the networks are constructed based on the previous definitions, and the degrees and costs of relationship have been calculated for linked nodes, we

¹ We will consider in this paper software artifacts at the file level, although others could be chosen, such as directories.

can apply standard SNA concepts to define a wide range of parameters of the network that can help characterising the network and its nodes [21]. Beyond classical parameters, others more sophisticated can be used to extract more specific information. This paper makes use of these parameters, calculated by means of Conan [14], which are introduced next:

- **Distance centrality** [25]: The distance centrality of a vertex, D_c , is a measurement of its proximity to the rest. It is sometimes called *closeness centrality* as the higher its value the closer that vertex is (on average) to the others. The distance centrality can be interpreted as a measurement of the influence of a vertex in a graph because the higher its value, the easier for that vertex to spread information through that network. Observe that when a given vertex is *far* from the others, it has a low degree of relationship (i.e. a high cost of relationship) with the rest.

Research has shown that employees who are central in networks learn faster, perform better and are more committed to the organisation. These employees are also less likely to turn over. Besides, from the point of view of information propagation, vertices with high centrality are like *hills* on the plain, in the sense that any knowledge is put on them is rapidly seen by the rest and spreads easily to the rest of the organisation.

- **Betweenness centrality** [8, 2]: The betweenness centrality of a vertex, B_c , is a measurement of the number of shortest paths traversing that particular vertex.

The betweenness centrality of a vertex can be interpreted as a measurement of the information control that it can perform on a graph, in the sense that vertices with a high value are intermediate nodes for the communication of the rest. In our context, given that we have weighted networks, multiple shortest paths between any pair of vertices are highly improbable. The betweenness centrality is just a measurement of the number of shortest paths traversing a given vertex.

In the SNA literature vertices with high betweenness centrality are known to cover *structural holes*. That is, those vertices glue together parts of the organisation that would be otherwise far away from each other. They receive a diverse combination of information available to no one else in the network and have therefore a higher probability of being involved in the knowledge generation processes.

- **Coordination degree** [1, 16]: The coordination degree of a vertex, measures the ability of this vertex in a graph to interchange information. The coordination degree of a vertex, has a great importance inside our study, since it shows the ability of a certain node to receive information of the network and capture the activity in a project precisely.

There are several manners to model this magnitude, but one of the easiest ways is to consider the coordination degree to be exponentially related to the distance between the vertices [16]. In this way, we define the coordination degree γ_{ij} between two vertices i and j as $\gamma_{ij} = e^{-\xi d_{ij}}$, where d_{ij} is the

distance between the two vertices and ξ is a real positive constant, measuring the strength of the relationship which we call the coordination strength.

Thus, we can define the total coordination degree of a vertex i in a graph as the sum of all the coordination degrees between that particular vertex and the rest. Namely, $\Gamma_i = \sum_{j=1}^N \gamma_{ij}$, where N is the order of the graph (the total number of vertices in that particular graph). The total coordination degree of a vertex is a measure of the amount of information that the vertex is able to receive belonging to that particular network.

- **Centrality Eigenvector** [4, 3]: Eigenvector centrality is a measure of the importance of a node in a network. It assigns relative scores to all nodes in the network based on the principle that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes.

Eigenvector centrality is defined as the principal eigenvector of the adjacency matrix defining the network. The defining equation of an eigenvector is $\lambda v = Av$ where A is the adjacency matrix of the graph, λ is a constant (the eigenvalue) and V is the eigenvector. The equation lends itself to the interpretation that a node that has a high eigenvector score is one that is adjacent to nodes that are themselves high scorers.

The idea is that even if a node influences just one other node, who subsequently influences many other nodes, then the first node in that chain is highly influential. Hence, the eigenvector centrality may give good advice about leadership in the project under study.

4 Case studies: Evolution and Mono

We have selected two FLOSS projects in order to apply the social network analysis techniques on them and to show what kind of information we can extract with such a type of analysis. In both projects, there is a high involvement of a company (Ximian) that has assigned several employees to the development of the project and there is a surrounding FLOSS community whose contributions are always welcome.

4.1 Evolution

Evolution is the official personal information manager and work group information management tool for GNOME. It combines e-mail, calendar, address book, and task list management functions.

The origin of the project goes back to 1998, when it was started by some GNOME volunteers. But it is not until a company called Helix Code identified it as a strategic project that it did not win in importance. In October 1999, Helix Code became Ximian Inc. due to trademark problems. Evolution, then, was mainly developed by Ximian employees although in strong collaboration with the GNOME community for several years. Finally, Ximian was acquired

by Novell in September 2004 and the maintenance of the then-mature Evolution tool was “outsourced” to Novell employees in India. Since then, contributions by the GNOME community are much lower, also in part because of the mature state of the tool.

There are other noteworthy events in the history of Evolution summarised in table 1.

Table 1. Most important events in the history of Evolution.

Event	Date
First commits	early 1998
Ximian takeover	October 1999
Version 0.0	July 2000
Version 1.0	February 2001
Takeover by Novell	October 2003
Death of main developer	January 2004
Version 2.0	October 2004
Version 2.2	April 2005
Version 2.4	October 2005
Version 2.6	April 2006
Version 2.8	October 2006

4.2 Mono

Mono is a project led by Novell (formerly by Ximian) to create an ECMA standard compliant .NET compatible set of tools, including among others a C# compiler and a Common Language Runtime. Mono can be run on Linux, FreeBSD, UNIX, Mac OS X, Solaris and Windows operating systems.

The dates of some of the most important events for Mono appear summarised in table 2.

5 Results

We have applied the SNA indexes to the two projects; results will be shown in this section. The samples have been taken from the first activity in each versioning system of the projects up to January 2007. Time slots of three months have been taken and two developers who have made at least one *commit* to any file within the same directory will be linked. This link will be weighted by the number of *commits* that both have in that time slot in the given directory. The interpretation of the following figures (each figure corresponds to a different measurement) is the following one: the horizontal axis is time, whereas the vertical z axis represents the result for a given measure. Therefore, each line in

Table 2. Most important events in the history of Mono.

Event	Date
First commits	mid-2001
First version of Gtk# checked into CVS	September 2001
Version 0.7	September 2001
Version 0.9	February 2002
GTK+ 2.2	December 2002
.NET Framework 1.1	April 2003
Version 1.0	June 2004
GNOME 2.10	March 2005
GTK+ 2.8	August 2005
.NET Framework 2.0	November 2005
Version 1.2	November 2006

the figure corresponds to the evolution over time of the values of a developer. We had to filter out less active developers as graphs that include all of them are hard to read. Hence, only the top 40 developers will be displayed.

5.1 Analysis of Evolution

If we analyse figure 1(a) representing the *coordination degree*, four phases can be clearly identified. An initial stage, until the takeover by *Ximian*, with a weak activity typical of a project in its early days. A second stage, until the *1.0 release*, where the work shows the highest gain in the history of the project, typical of the effort for developing a first stable version. Since then and until the takeover by *Novell*, a significant stagnation is originated due to a maintenance and debugging period. Finally, and because of the fact that a periodic release policy is introduced, peaks of work of two different dimensions can be observed alternating in time. The highest peaks correspond to new releases, with a high number of commits corresponding to new developments, translations and documentation. Meanwhile, the lows are due to stages of development and maintenance in which the kind of activity is different, being reflected in a smaller number of commits. It is therefore significant that time-based release management (i.e. a new version will be released every 6 months) seems to boost the development in general terms in comparison to feature-based release management (a new version will be made public when all features to be included are finished).

In addition to the own activity of a project, the *average coordination degree* of a network provides a measurement of the efficiency if we compare it with other values, such as the number of commits or the number of active developers. Figure 1(b) provides this comparison.

If we pay attention to the curves of the *average coordination degree* (cd mean) and the number of active developers (nodes), we can notice different moments at which one is over the other. The interpretation of these relative

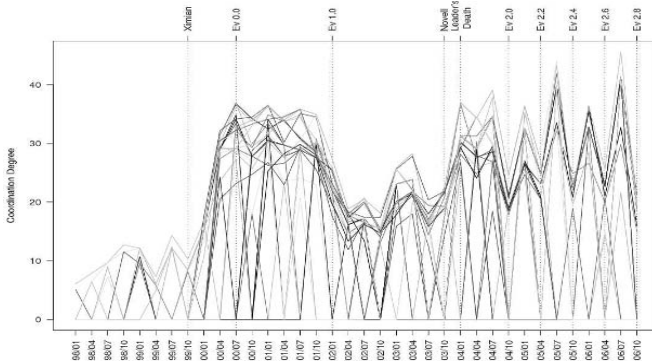
positions is the following: when the average coordination degree of the network is over the number of developers, the network structure is efficient. However, when the curve of the number of developers is over the average coordination degree, the social structure of the network is not efficient. In this way we can observe two stages with low efficiency in the network, one of them from the beginning of the project until the takeover by *Ximian*, the other one from the time when periodic releases were introduced (2004) until today. In between, since the takeover by *Ximian* in 1999 and up to the release of version *2.0* (in 2004), the network has an efficient structure. This evidences that a corporate involvement in the development of a FLOSS project may result in a more efficient development, but that both the involved company and the community have to find ways to allow the other party to participate properly. The first phase, where there was no involvement of the company, or the last one, where it seems that the company has chosen to move away from community have lead to an inefficient structure.

Figure 2(a) gives the evolution over time of the *betweenness centrality* for each developer. This figure is useful to identify who is the leader of the project at any given time, but also allows to predict who could be the following leader. If we attend the leadership curve in a particular moment, we can look for the curve just underneath. Usually, as the leader succession takes place gradually, we could figure out who is going to be the next leader of the project if the current one reduces his activity substantially.

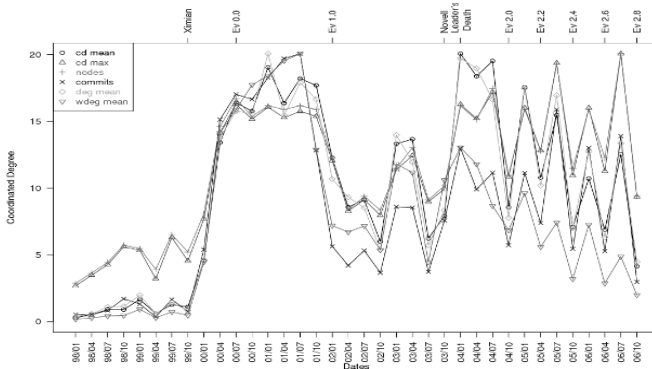
Figure 2(b) is a bar chart with the main leaders of the project for every time period. The measure that has been used is the *weighted betweenness* (the same as in Figure 2(a)). For the computation of this measure, we have to count the shortest paths that cross a vertex, so this measure is dependant on the number of developers in the network. For this reason, although the maximum values evolve at the same pace as the number of developers grows, it is an efficient measurement. From these results, three different stages can be observed. The two first stages in which "unammx" and "ettore" lead the project during several years, and one third stage since mid-2003 in which different leaders succeed themselves in short periods of time (all of them less than one year).

As noted before, the *eigenvalue* can be interpreted like a numerical assignment of the relevance of a node in the network. According to this interpretation, the higher the value of a given node, the higher its relevance. In figure 3(a) we can observe the evolution over time of this measure for each developer. At every moment, we can identify the main leader(s) of the project. If we compare the outline of the curve with the *coordination degree* or with the values of *betweenness*, it is possible to notice that the values of this measure do not depend on the number of developers, and therefore, the activity of the project is not reflected in the outline of the curves.

If we attend to the figure 3(b), we can find again the main leaders in every time slot, this time using the *eigenvalue*. If we compare these results with the ones obtained for the *betweenness*, we will see high similarities as expected. However, we must emphasise the differences in some time slots in which the



(a) Evolution of the Coordination Degree for Evolution



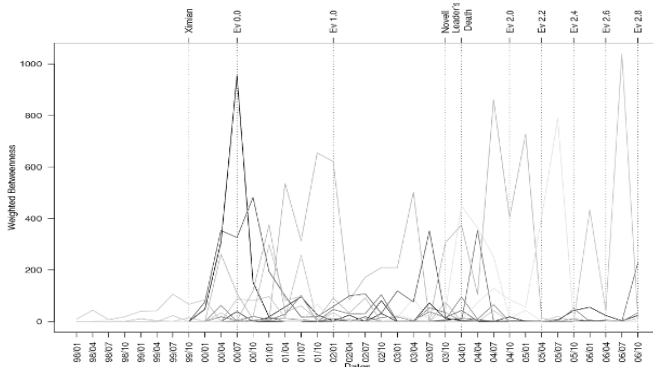
(b) Coordination Degree comparative for Evolution

Fig. 1. Coordination Degree values and a comparative for top 40 committers in Evolution.

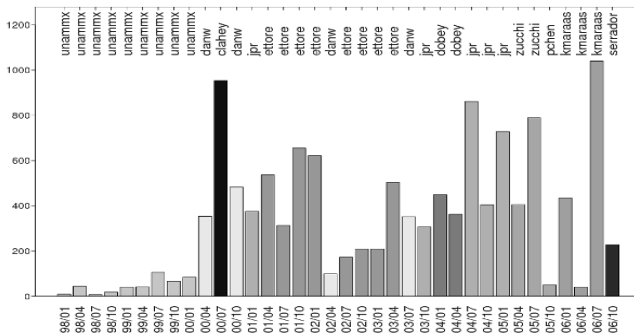
identified leader differs. This is because the *betweenness* focuses on the control of the information flows whereas the *eigenvalue* provides a node valuation from the point of view of centrality. Thus, the periods where leaders remain the main contributors to a project are smaller for the second value, because a greater effort has to be carried out to keep this privileged position. In the same way, the number of different persons who achieve the leadership is higher using *eigenvalues*. So, we can see that gaining control over information flows in the project is an easier task than obtaining a high influence on the network.

5.2 Analysis of Mono

The activity of Mono is marked clearly by five events that can be appreciated in Figure 4(a). The first one is the *Mono 0.7* release (first unstable version) that additionally happens at the same time as the initial check-in of the *GTK#*



(a) Evolution of the Weighted Betweenness for Evolution

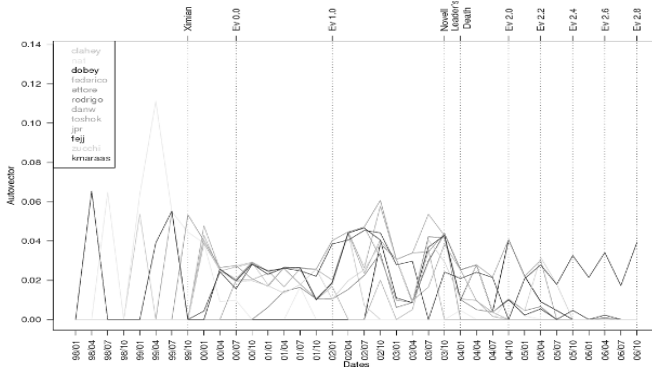


(b) Weighted Betweenness comparative for Evolution

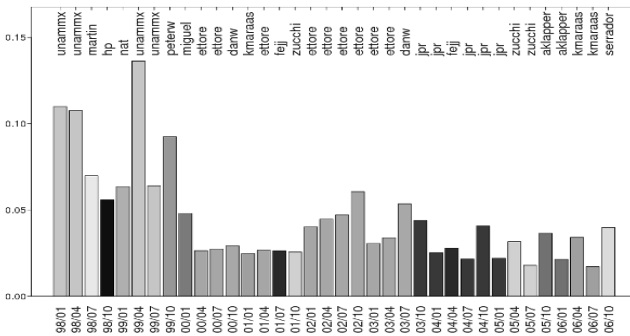
Fig. 2. Weighted Betweenness values and a comparative for top 40 committers in Evolution.

module in the repository. Then a progressive increase of the activity took place, reaching its highest peak in the middle of 2003. An external fact caused an extraordinary activity: the *.NET 1.1* release, which Mono had to react to be able to support this specification in its next version. This adaptation was completed in *Mono 1.0* in 2004. In 2005 another noteworthy moment took place, when the Mono team started to adapt the *.NET 2.0* specifications.

The development of Mono does not have a continuous growth, but it shows momentary efforts to achieve the *.NET* functionality. This kind of development is reflected in its social structure as it is not efficient most of the time. As it can be observed from Figure 4(b) while the number of developers grows smoothly, their activity has several peaks. In any case, compared to the previous case study, we can conclude that in this case Ximian has not achieved such a high community involvement in Mono as in Evolution.



(a) Evolution of the Eigenvalue for Evolution

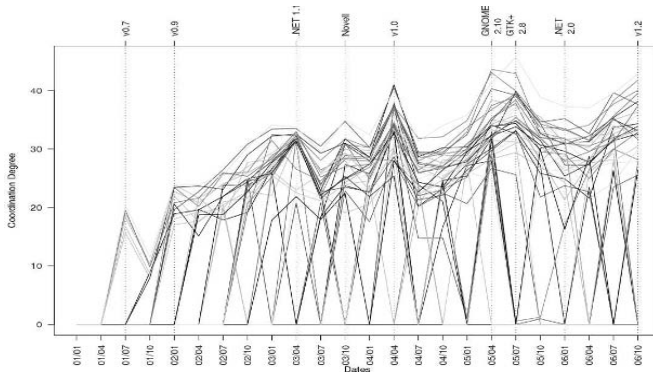


(b) Eigenvalue comparative for Evolution

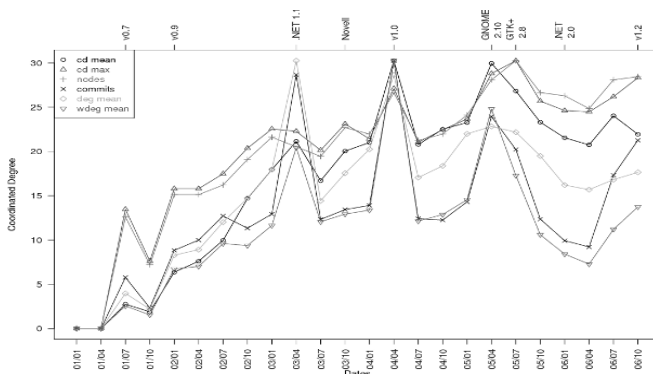
Fig. 3. Eigenvalue values and a comparative for top 40 committers in Evolution.

The Mono project is composed currently of over 80 developers. As it can be seen from Figure 5(a), and a large difference between the values of the main leader and the ones for the rest of developers exist. The yellow line belongs to Miguel de Icaza, the founder of the project, while the red one to Gonzalo Paniagua, a Novell-hired developer, and the green lines correspond to Raja R. Harinath and Ben Maurer, also Novell employees. Miguel de Icaza is clearly the main leader in Mono as he has the maximum value in 16 quarters, in the early stages and also recently. Gonzalo Paniagua had a more prominent role in the first half of the project as well as in the last phases.

If we use *eigenvalues* (see Figure 6(a)), the difference between the project leaders and the rest of developers are not that high as found using *betweenness*. In any case, if we compare Figure 6(b) with figure 5(b) the main leaders continue being the same. Even though, eigenvalue helps us obtaining additional information on the development community as it allows to rank developers by their activity and their position in the network.



(a) Evolution of the Coordination Degree for Mono



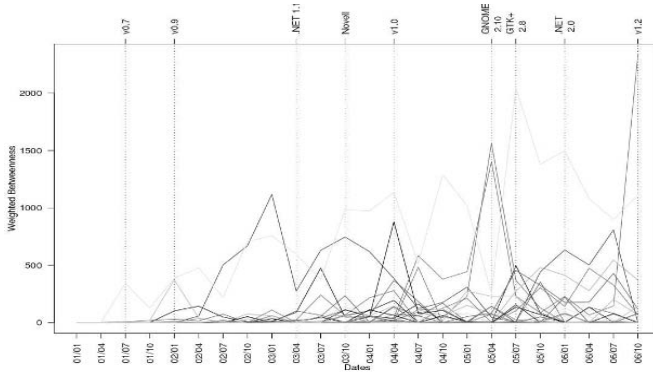
(b) Coordination Degree comparative for Mono

Fig. 4. Coordination Degree values and a comparative for top 40 committers in Mono.

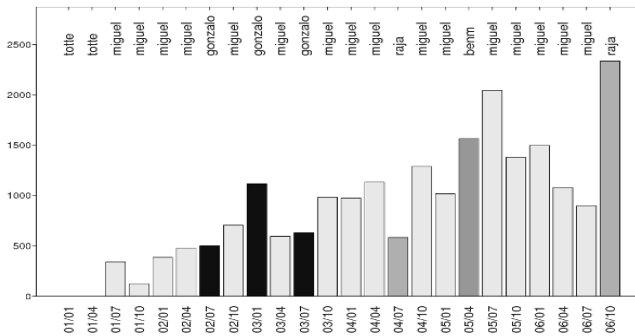
6 Conclusions and Further work

In this paper a descriptive study of several social networks of FLOSS projects where a company and the community collaborate has been presented. We have seen that although there is access to the interactions that developers have in the source code management system, the amount of information is that high that we require additional methods to properly analyse the community and extract facts and information from it.

Thanks to the use of social networks analysis techniques we have been able to obtain some information on these projects, in particular, that time-based release management seems to animate the development of projects or that the collaboration between the company and the community can, if properly handled, drive a more efficient development. Future research should be devoted to analyse the differences between the studied projects to try to find out the aspects that drive one of the case studies (Evolution) to have at least for some



(a) Evolution of the Weighted Betweenness for Mono



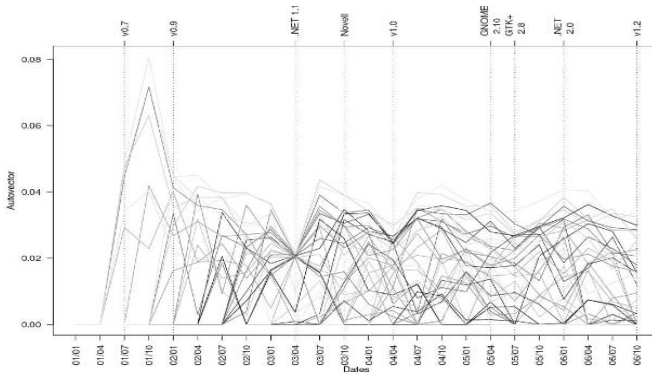
(b) Weighted Betweenness comparative for Mono

Fig. 5. Weighted Betweenness values and a comparative for top 40 committers in Mono.

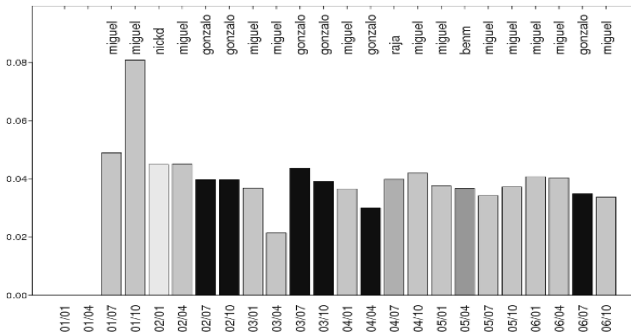
time an efficient development, while this is an exception for the other project (Mono) under study.

In addition, we have identified the most important nodes (developers) in the project and have observed them from from several points of view (leadership, information control flows, etc.) and how they evolve over time. In both case studies, the most prominent positions were held by company employees, showing a low turnover over the years. Future work could be devoted to find out if this behaviour is similar in community-led projects.

All in all, this paper demonstrates that the techniques from the social networks analysis field can be of great interest for the study and characterisation of any kind of network and with particular interest in the study of how the software industry can interact with the FLOSS community entering the development of projects.



(a) Evolution of the Eigenvalue for Mono



(b) Eigenvalue comparative for Mono

Fig. 6. Eigenvalue values and a comparative for top 40 committers in Mono.

7 Acknowledgements

This work has been funded in part by the European Commission, under the QUALIPSO (FP6-IST-034763), FLOSSMETRICS (FP6-IST-5-033547) and QUALOSS (FP6-IST-5-033547) projects, and by the Spanish CICyT, project SobreSalto (TIN2007-66172).

We would like to thank the QualiPSo Activity 6 partners for comments and suggestions on this research.

References

1. J. A. Almendral, L. Lopez, and M. A. F. Sanjuan. Information flow in generalized hierarchical networks. *Physica A Statistical Mechanics and its Applications*, 324:424–429, June 2003.

2. Jac M. Anthonisse. The rush in a directed graph. Technical report, Stichting Mathematisch Centrum, Amsterdam, The Netherlands, 1971.
3. Phillip Bonacich. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology*, 2:113–120, 1972.
4. Stephen P. Borgatti. Centrality and network flow. *Social Networks*, 27(1):55–71, January 2005.
5. Kevin Crowston and James Howison. The social structure of free and open source software development. *First Monday*, 10(2), February 2005.
6. Jean-Michel Dalle and Paul A. David. The allocation of software development resources in open source production mode. Technical Report SIEPR Discussion Paper No. 02-27, Stanford Institute for Economic Policy Research, February 2003.
7. Trung T. Dinh-Trong and James M. Bieman. The FreeBSD project: A replication case study of Open Source development. *IEEE Transactions on Software Engineering*, 31(6):481–494, June 2005.
8. Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry* 40, 35–41, 1977.
9. Daniel M. German. The GNOME project: a case study of open source, global software development. *Journal of Software Process: Improvement and Practice*, 8(4):201–215, 2004.
10. Roger Guimera, Albert Diaz-Guilera, F. Vega-Redondo, A. Cabrales, and Alex Arenas. Optimal network topologies for local search with congestion. *Physical Review Let.* 89, 248701, 2002.
11. Israel Herraiz, Gregorio Robles, Juan José Amor, Teófilo Romera, and Jesús M. González Barahona. The processes of joining in global distributed software projects. In *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 27–33, New York, NY, USA, 2006. ACM Press.
12. Chris Jensen and Walter Scacchi. Modeling recruitment and role migration processes in OSSD projects. In *Proceedings of 6th International Workshop on Software Process Simulation and Modeling*, St. Louis, May 2005.
13. Stefan Koch and Georg Schneider. Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal*, 12(1):27–42, 2002.
14. Luis Lopez and Juan Martinez-Romo. Conan: Generating social network analysis. <http://tools.libresoft.es/conan>.
15. Luis Lopez, Gregorio Robles, Jesus M. Gonzalez Barahona, and Israel Herraiz. Applying social network analysis techniques to community-driven libre software projects. *International Journal of Information Technology and Web Engineering*, 1(3):27–48, July–September 2006.
16. Luis Lopez and Miguel A. F. Sanjuan. Relation between structure and size in social networks. *Phys. Rev. E*, 65(3):036107, Feb 2002.
17. Gregory Madey, Vincent Freeh, and Renee Tynan. Modeling the Free/Open Source software community: A quantitative investigation. In Stefan Koch and Stefan Koch, editors, *Free/Open Source Software Development*, pages 203–221. Idea Group Publishing, Hershey, Pennsylvania, USA, 2004.
18. Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of Open Source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

19. Audris Mockus and Lawrence G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance*, pages 120–130, October 2000.
20. Mark E. J. Newman. Scientific collaboration networks: I. network construction and fundamental results. *Phys. Rev. E* 64, 016131, 2001.
21. Gregorio Robles. *Empirical Software Engineering Research on Libre Software: Data Sources, Methodologies and Results*. PhD thesis, Escuela Superior de Ciencias Experimentales y Tecnología, Universidad Rey Juan Carlos, 2006.
22. Gregorio Robles, Jesus M. Gonzalez-Barahona, and Martin Michlmayr. Evolution of volunteer participation in libre software projects: evidence from Debian. In *Proceedings of the 1st International Conference on Open Source Systems*, pages 100–107, Genoa, Italy, July 2005.
23. Gregorio Robles, Stefan Koch, and Jesus M. Gonzalez-Barahona. Remote analysis and measurement of libre software systems by means of the CVSAly tool. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, pages 51–56, Edinburgh, Scotland, UK, 2004.
24. Gregorio Robles, Juan Julian Merelo, and Jesus M. Gonzalez-Barahona. Self-organized development in libre software: a model based on the stigmergy concept. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling (ProSim 2005)*, St.Louis, Missouri, USA, May 2005.
25. Gert Sabidussi. The centrality index of a graph. *Psychometrika* 31, 581-606, 1996.
26. Yuwan Ye, Kumiyo Nakakoji, Yasuhiro Yamamoto, and Kouichi Kishida. The co-evolution of systems and communities in Free and Open Source software development. In Stefan Koch and Stefan Koch, editors, *Free/Open Source Software Development*, pages 59–82. Idea Group Publishing, Hershey, Pennsylvania, USA, 2004.
27. Thomas Zimmermann, Peter Weissgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, June 2005.

Empirical Analysis of the Bug Fixing Process in Open Source Projects

Chiara Francalanci and Francesco Merlo
Dipartimento di Elettronica ed Informazione, Politecnico di Milano
via Ponzio 34/5, I-20133 Milano, Italy
{francala | merlo}@elet.polimi.it

Abstract. Monitoring the performance of processes is often considered critical in classic engineering fields. However, in the area of software engineering (and especially in the Open Source context) it seems that the literature has not yet taken into consideration the problem of identifying the process characteristics and performance of debugging. The aim of this paper is the identification of the performance characteristics of the bug fixing process of Open Source applications, focusing on continuity and efficiency indicators. The importance of such indicators is even more relevant today, since Open Source software is now adopted also in many business contexts. We have analyzed the debugging process of 9 active and popular Open Source projects, collecting a dataset comprising more than 65,000 closed bugs. Results have highlighted four types of bug fixing processes that can be distinguished by considering temporal continuity and efficiency dimensions.

Key Words: Open Source Software debugging process quality

1 Introduction

Debugging is defined in [9] as “the activity of locating and correcting errors” in software programs. It is an important part of the software development and maintenance process, since a lot of time is spent by developers in activities related to the location and resolution of bugs. As pointed out by a NIST research [21], software bugs are so prevalent and so detrimental that they cost the U.S. economy an estimated \$59.5 billion annually, or about 0.6 percent of the gross domestic product.

In the literature, many studies have focused on debugging. Relevant results have been achieved in the field of software testing, with the aim of detecting the presence of errors in programs. An overview of software testing can be found in [16, 2, 13, 9]. However, as noted by [9], debugging is “one of the least understood activities in software development and is practiced with the least amount of discipline; it is often approached with much hope and little planning”. In particular, one of the most difficult problems is bug localization, that is, finding where bugs are located in the

Please use the following format when citing this chapter:

Francalanci, C. and Merlo, F., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 187–196.

source code (as opposed to testing, which is concerned with the identification of the presence of bugs). Consequently, bug localization is one of the most widely addressed problems. Techniques such as statistical debugging [20, 12, 25, 17, 1] try to identify bug predictors (e.g., software quality metrics like CK object-oriented metrics [5]) that may refer to the actual bug location.

Although some studies have already been published [15, 6, 19, 14, 7], it seems that the problem of identifying the process characteristics and performance of debugging has not been taken into account yet.

For other “classic” engineering processes, performance indicators are considered critical. The importance of such indicators is even more relevant today since Open Source software is now adopted also in many business contexts. As noted by Wasserman and Capra [24], Goldman [10], Goth [11] and Riehle [18], when evaluating an Open Source product for business adoption a number of variables different from those considered for the classical proprietary software should be taken into account. For example, the quality of the support provided by the community, or the quality of the bug fixing process itself are cited as new software selection variables to be taken into account in the Open Source context.

The aim of this paper is the identification of the performance characteristics of the bug fixing process of Open Source applications that can be used as indicators to assess the quality of the process itself. In particular, our goal is to focus on variables related to the continuity and efficiency of the bug fixing process, since these characteristics are among the most relevant when assessing debugging performance, especially in a business context.

The paper is structured as follows: Section 2 describes the data sample we have used for the analysis of bug fixing processes, Section 3 presents the analysis and the related empirical results, while Section 4 discusses the main findings and presents some concluding remarks on this work.

2 Data Sample

The dataset used for this study is composed by 9 active and popular Open Source projects: Table 1 shows the summary descriptive statistics of the dataset. Each application is identified by a mnemonic code.

A tool developed *ad-hoc* has been used to collect data. In particular, the tool has been used to parse the information provided by the bug tracker of each project. In order to guarantee the homogeneity of the collected dataset, we have used the same parameters to query each bug tracker. That is, we have focused on bugs with closed status related only to Microsoft Windows and Linux operating systems. Moreover, we have excluded all external components or additional plugins, analyzing only bugs located in the core components of each project (the project sizes reported in Table 1 are however related to the whole applications).

Table 1. Dataset summary statistics.

Mnemonic Code	Domain	Size (KSLOC)	Collected Bugs	Time span (months)	Bug tracking Tool
A	IDE	2,063	51,485	65.4	Apache's JIRA
B	IDE	152	2,529	76.6	SF Tracker Tool
C	DBMS	713	8,478	48.5	bugs.php.net Bugs Sys
D	DBMS	810	997	26.2	Apache's JIRA
E	DBMS	660	577	78.8	SF Tracker Tool
F	DBMS	159	737	65.9	SF Tracker Tool
G	DBMS	392	515	34.0	SF Tracker Tool
H	DBMS	522	270	65.8	Apache's JIRA
I	DBMS	153	713	8.0	Apache's JIRA
Total		5,624	66,301	ns	
Average		624	7,367	52.1	

The selection of bugs related to the core components of each application has been performed by manually specifying the correct parameters when querying the bug tracking systems. After the data collection phase, a manual data preprocessing has been carried out to ensure data cleaning and normalization. For each bug, we have collected information about its status (either *opened* or *closed*), the opening and closing dates, the affected project component, the level of priority/severity, the textual description, the application version affected and the operating system.

3 Empirical Analyses and Results

This section presents the analyses that we have performed on our dataset and related empirical results. In particular, we have focused our analyses on the time required to close bugs (referred to as MTTR - Mean Time To Repair, by analogy with hardware systems), defined for a generic bug b as follows:

$$MTTR(b) = date_{closed}(b) - date_{opened}(b),$$

where $date_{closed}(b)$ and $date_{opened}(b)$ refer to the change of bug status (*closed* and *open*, respectively) registered by the project bug tracker tool. As a consequence, we have focused our analyses only on *closed bugs*.

The influence of the release train mechanism on the bug fixing process. The first analysis that we have carried out on the dataset has been a qualitative evaluation of the MTTR values for each project. To accomplish this task, we have considered for each application in our dataset the scatter plot of the MTTR values of each bug collected by the bug tracker. Figure 1 shows an example for project C. Each data point in the plot identifies a closed bug; the X axis reports the bug's opening date, while the Y axis reports the MTTR value.

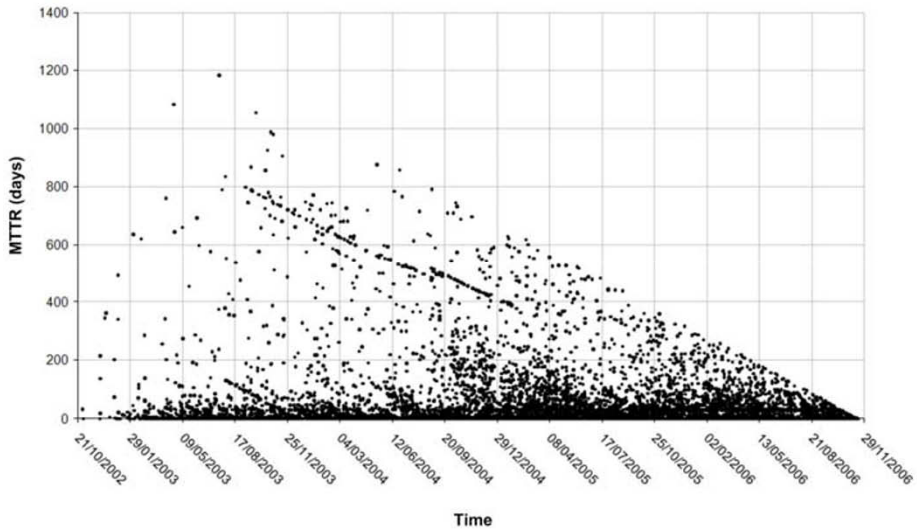


Fig. 1. Scatter plot of MTTR values of project C.

It is worth noting that Figure 2 (the scatter plot of MTTR values of project F) shows a set of oblique lines: these lines can be interpreted as an effect of the release train mechanism, since each line includes the bugs that have been closed in the proximity of a release date. The intuitive meaning of this phenomenon is that when a release date approaches, developers hasten to correct bugs with the aim of including corrections in the forthcoming release. In fact, if changes are verified and committed before the scheduled deadline, they can be included in the release; otherwise, they would shift to the next release. Figure 2 shows how each line can be traced back to the release date of a version of the application. By checking the dates identified on the plots against the official release dates taken from the website of each project, we have verified that more than 90% of official release dates could be correctly identified in this way.

From the bug fixing quality point of view, this phenomenon is certainly relevant. If bug corrections are made too close to the release date, the overall quality of the bug fixing process can be worse, due to the possible inaccuracy of interventions and scarce documentation of changes. As noted by Capra et al. [4], these are some of the most relevant causes that contribute to increase software entropy, which is proved to negatively affect overall software quality and increase maintenance costs.

The bug fixing process quality continuum. A fundamental point that has to be taken into account when considering the quality of the bug fixing process is the analysis of the bug opening and closing trends.

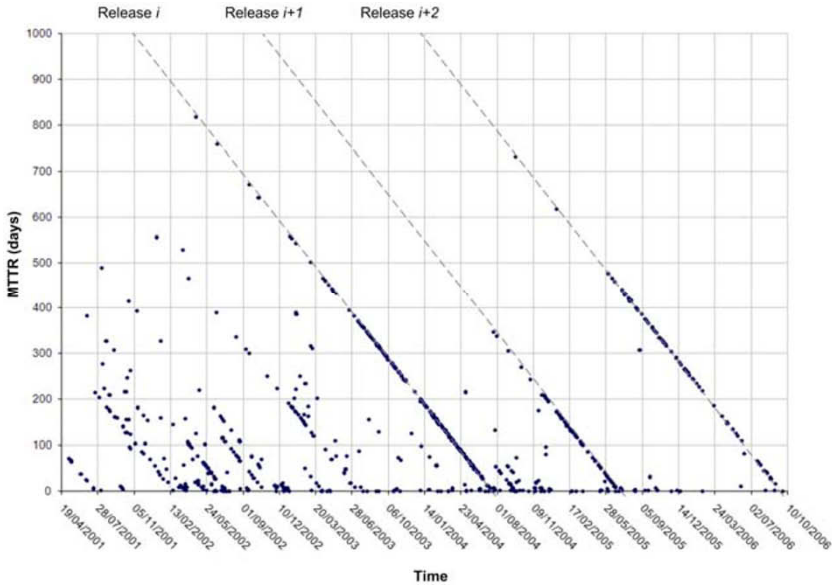


Fig. 2. Influence of the release train mechanism on bug fixing process of project F.

The bug opening trend is related to the process of submission and acceptance of new bugs by considering the cumulated number of opened and verified bugs over time. The bug closing trend is the cumulated number of bugs that are resolved and closed over time. Figure 3 shows an example of bug opening and closing trend curves. As it can be noted, the distance between the two curves at a given point in time represents the number of bugs simultaneously open at that time. The closing trend curve can be directly derived from the distribution of MTTR values: for each time interval (typically on a daily basis), the increment of the closing trend curve is equal to the number of closed bugs in that time interval.

A high quality debugging process is generally considered to be continuous over time [23]. That is, there should not be any discontinuity in the debugging process and the number of closed bugs should grow at least as fast as the number of opened bugs to avoid the uncontrolled growth of unresolved bugs.

The characteristics of the ideal bug fixing process can be translated into visual properties of the bug opening and closing trend curves:

- Process continuity - trend smoothness: if the bug fixing process is continuous, then the closing trend curve is smooth and without peaks or steps.
- Process efficiency - number of open bugs: if the debugging process is efficient (i.e., bugs are closed at least at the same rate with which they are opened), the closing trend curve stays near to the opening trend curve, without increasing the overall number of open bugs.

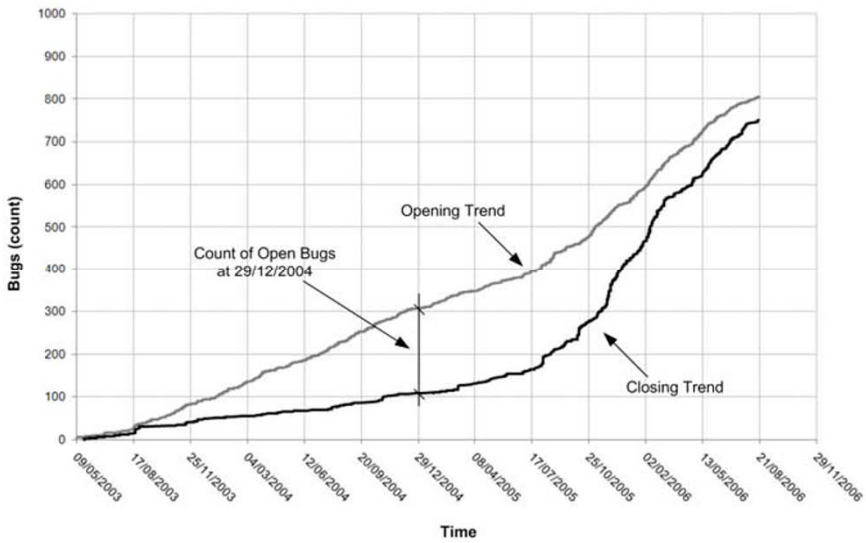


Fig. 3. Bug opening and closing trends.

Based on these considerations, it is possible to assess different quality levels of the bug fixing process by analyzing the bug opening and closing trend curves. It is worth noting that, at this point, evaluations are qualitative. Figure 4 shows four examples of trend curves for projects in our dataset, ordered by the quality level of the bug fixing process. As it can be noted, a high variance of the quality level of the bug fixing process can be identified among the projects in our dataset.

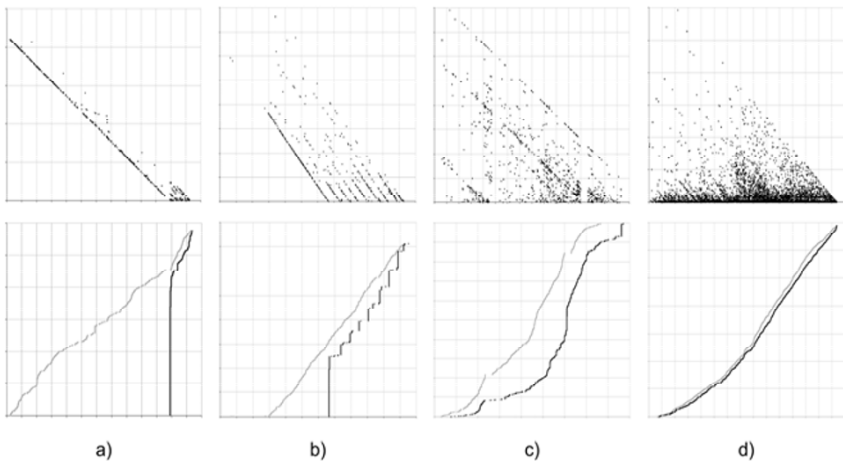


Fig. 4. Sample bug fixing processes: a) project E; b) project I; c) project D; d) project B.

Case *a)* of Figure 4 denotes a low quality bug fixing process since the closing trend shows a step, corresponding to a mass correction of bugs. At the opposite end, case *d)* can be considered as a high quality bug fixing process: the closing trend curve is smooth and very close to the opening trend curve. Cases *b)* and *c)* are positioned along the quality continuum between these two extremes: case *b)* represents a discontinuous process since the closing curve shows numerous steps, while case *c)* is a more continuous but not efficient process, since the closing trend curve is quite far from the opening trend curve. Please note that these considerations are not influenced by the fact that in practice there might be some bugs that are never fixed, as noted for example by [7]: since this seems to be a very common problem [14, 7, 22], it can be considered a common bias uniformly spread across projects.

The projects in our dataset can thus be divided into four categories on the basis of the visual properties of the opening and closing curves:

- a) Inefficient and discontinuous;
- b) Inefficient and continuous;
- c) Efficient and discontinuous;
- d) Efficient and continuous.

Distinctive properties of the bug fixing process. The distinction between different levels of quality of the debugging process should be quantified. Along with the scatter plot we have considered also the distribution of MTTR values. Figure 5 shows two examples related to projects B and E. This kind of graph shows the number of bugs that have been closed with the same MTTR value. By analyzing the distribution of the MTTR values of a project, two statistical indexes can be considered to quantify the quality level of the debugging process: skewness and kurtosis. The values of such indexes can provide significant information to compare different bug fixing processes, given that the MTTR values are comparable.

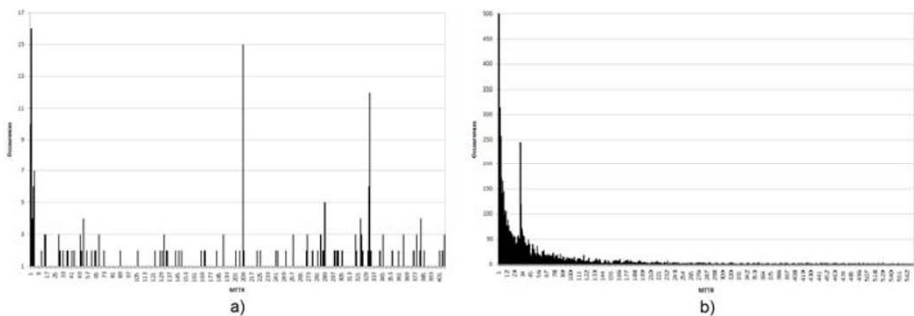


Fig. 5. Distribution of MTTR values for *a)* project E and *b)* project B.

The skewness index γ is a measure of the asymmetry of the distribution of the MTTR. That is, the distribution shows different tail shapes on the two sides of the average value. Considering the debugging process, the skew is typically positive, since it is more frequent that the MTTR distribution is right-tailed: in fact, the mean value of MTTR is typically influenced by a number of bugs whose closing times are

much higher than the average one (i.e., a high value of skewness can be considered as indicator of the presence of “superbugs”, see [7]).

The kurtosis index κ is a measure of the *peakedness* of the distribution of the MTTR. Higher kurtosis values indicate that most of the variance is due to a few very high deviations, as opposed to frequent small deviations from the mean value of MTTR. From the bug fixing process point of view, a high kurtosis means that the variance of the MTTR distribution can be explained by a few bugs that have required extremely long closing times. As a consequence, high kurtosis values are desirable, since they can be considered as an index of efficiency. For instance, by considering case *a*) of Figure 5, it is clear that many bugs show a closing time much higher than the average MTTR value, while in case *b*) the number of bugs with very high MTTR is considerably lower and negligible with respect to case *a*).

4 Conclusions

Results indicate that the release train mechanism, widely adopted in the Open Source context, affects the debugging process. In particular, there is empirical evidence that a great number of bugs tends to be resolved close to scheduled releases. From a project management point of view, this phenomenon should be taken into account when scheduling releases, since if they are too frequent the quality of interventions for bug resolution could be lowered. Consequently, software entropy could be increased, leading to higher maintenance costs/effort. From the user point of view, a project with a debugging process that is too affected by the release train mechanism could be less dependable due to inaccurate bug corrections.

Another relevant empirical result is that even in the Open Source context, which is commonly addressed to as a high quality software development practice [8, 3], the quality of the bug fixing process is extremely variable. The projects in our dataset showed different quality levels, which can be positioned along a continuum depending on the performance of the bug fixing process.

The quantitative distinction between different bug fixing quality levels can be measured by means of two statistical indexes of the MTTR distribution, namely skewness and kurtosis. These indexes measure two relevant characteristics of the debugging process, temporal continuity and efficiency.

Future work is focused at extending the empirical data set of applications to provide better statistical significance and at better understanding the skewness and kurtosis indexes from a software development process point of view.

Acknowledgements

We would like to thank Marco Balzarini for his support in the data collection and analysis phases.

References

1. The cooperative bug isolation project. <http://www.cs.wisc.edu/cbi>.
2. B. Beizer. *Software Testing Techniques*. John Wiley and Sons, New York, 1983.
3. Boulanger. Open source versus proprietary software: is one more reliable and secure than the other? *IBM Systems Journal*, vol. 44, no.2:p. 239, 2005.
4. E. Capra, C. Francalanci, and F. Merlo. The economics of open source software: An empirical analysis of maintenance costs. In *Proc. Int'l Conf. Software Maintenance*, pages 395-404, 2007.
5. S. Chidamber and C. Kemerer. A metrics suite for object-oriented design. *IEEE Trans. Software Eng.*, vol.20, no. 6:pp. 476-493, 1994.
6. K. Crowston and B. Scozzi. Coordination practices for bug fixing within FLOSS development teams. In *Proceedings of CSAC*, 2004.
7. J. M. Dalle and M. den Besten. Different bug fixing regimes? a preliminary case for superbugs. In *Proceedings of OSS Systems Conference*, pages 247-252, 2007.
8. Fitzgerald. A critical look at open source. *IEEE Computer*, vol. 37, no. 7:pp.92-94, 2004.
9. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, New Jersey, 2003.
10. R. Goldman and R. Gabriel. *Innovation happens elsewhere: Open Source as business strategy*. Morgan Kauffmann, 2005.
11. G. Goth. Open source business models: ready for prime time. *IEEE Software*, vol. 22, no. 3:pp. 99-100, 2005.
12. T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Trans. Software Eng.*, vol. 31, no. 10:pp. 897-910, 2005.
13. W. E. Howden. *Functional program testing*. McGraw Hill, New York, 1987.
14. M. Michlmayr and A. Senyard. *The Economics of Open Source Software Development*, chapter A Statistical analysis of defects in Debian and strategies for improving quality in free software projects, pages 131-148. Elsevier B. V., 2006.
15. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development. *ACM Transactions on Software Engineering Methodology*, vol. 11:pp. 309-346, 2002.
16. G. J. Myers. *The Art of Software Testing*. John Wiley and Sons, New York, 1979.
17. P. A. Nainar, T. Chen, J. Rosin, and B. Liblit. Statistical debugging using compound boolean predicates. In *Proc. Int'l Symp. Software Testing and Analysis*, 2007.
18. Riehle. The economic motivation of open source software: stakeholder perspective. *IEEE Computer*, vol.40, no. 4:pp. 25-32, 2007.
19. R. J. Sandusky, L. Gasser, and G. Ripoché. Bug report networks. In *Proceedings of ICSE MSR*, 2004.
20. R. Subramanyam and M. S. Krishnan. Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects. *IEEE Trans. Software Eng.*, vol. 29, no. 4:pp. 297-310, 2003.
21. G. Tasse. The economic impacts of inadequate infrastructure for software testing. Technical Report 7007.011, National Institute of Standards and Technology, Acquisition and Assistance Division, May 2002.
22. L. Villa. Large free software projects and Bugzilla: Lessons from GNOME project QA. In *Proceedings of the Linux Symposium*, 2003.

23. H. Wang and C. Wang. Open source software adoption: a status report. *IEEE Software*, vol. 18; no. 2:pp. 90-95, 2001.
24. I. Wasserman and E. Capra. Evaluating software engineering processes in commercial and community open source. In *Int'l Workshop Emerging Trends in FLOSS Research and Development*, 2007.
25. Y. Zhou and H. Leung. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Trans. Software Eng.*, vol. 32, no. 10:pp. 771-789, 2006.

The Total Growth of Open Source

Amit Deshpande
SAP Research, SAP Labs LLC
3475 Deer Creek Rd
Palo Alto, CA, 94304 U.S.A.
amit@amit-deshpande.com

Dirk Riehle
SAP Research, SAP Labs LLC
3475 Deer Creek Rd
Palo Alto, CA, 94304 U.S.A.
dirk@riehle.org

Abstract. Software development is undergoing a major change away from a fully closed software process towards a process that incorporates open source software in products and services. Just how significant is that change? To answer this question we need to look at the overall growth of open source as well as its growth rate. In this paper, we quantitatively analyze the growth of more than 5000 active and popular open source software projects. We show that the total amount of source code as well as the total number of open source projects is growing at an exponential rate. Previous research showed linear and quadratic growth in lines of source code of individual open source projects. Our work shows that open source is expanding into new domains and applications at an exponential rate.

1 Introduction

Software development is undergoing a major change from being a fully closed software development process towards a more community driven open source software development process. Successful open source projects like Linux, Apache, PostgreSQL and many others are growing super-linearly. Previous research showed that linear and quadratic growth is the dominant growth pattern of open source software projects [5] [8] [15] [16] [18] [22].

In this paper, we analyze the combined growth of open source software in terms of lines of source code as well as number of projects. Our database contains more than 5000 active and popular open source projects. The database provides fine granular data of developer actions over the last 17 years from 1990 to 2006. We analyze the average amount of source code added per month for the time frame of January 1995 to December 2006 as well as the number of projects added over time.

We find that both the growth rate as well as the absolute amount of source code is best explained using an exponential model. Given that previous research showed that most open source projects grow at a polynomial rate, we suggest and then verify that the number of open source projects is growing at an exponential rate.

This paper is organized as follows. Section 2 discusses our motivation, the hypothesis, and its implications. Section 3 discusses our database and approach. Section 4 presents the results of the analysis. Section 5 discusses some limitations of the analysis and Section 6 discusses related work. Section 7 concludes the paper.

Please use the following format when citing this chapter:

Deshpande, A. and Riehle, D., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 197–209.

2 The Growth of Open Source

Open source software is having a major impact on the software industry and its production processes. Many software products today contain at least some open source software components. Some commercial products are completely open source software [9]. In some markets, for example, web servers, open source software holds a dominant market share [10].

Open source software today has a strong presence in industry and government. Walli *et al.* observe [19]: “Organizations are saving millions of dollars on IT by using open source software. In 2004, open source software saved large companies (with annual revenue of over \$1 billion) an average of \$3.3 million. Medium-sized companies (between \$50 million and \$1 billion in annual revenue) saved an average \$1.1 million. Firms with revenues under \$50 million saved an average \$520,000.”

Commercially, the significance and growth of open source is measured in terms of revenue generated from it. Lawton and Notarfonzo state that packaged open source applications generated revenues of \$1.8 billion in 2006 [9]. The software division of the Software & Information Industry Association estimates that total packaged software revenues were \$235 billion in 2006 [4]. Thus, open source revenue, while still small compared to the overall market (~0.7%) is not trivial any longer.

However, open source software today is part of many proprietary (closed) source products, and measuring its growth solely by packaged software revenue is likely to underestimate its size and growth by a wide margin. To measure the growth of open source we need to look at the total growth of open source projects and their source code.

Several studies have been undertaken to measure the growth and evolution of individual open source software projects [5] [15] [16] [18]. Most of these studies are exemplary, focusing on a few selected projects only. The exception is Koch’s work, which uses a large sample (>4000 projects) to determine overall growth patterns in open source projects, concluding that polynomial growth patterns provide good models for these projects [8] [20]. Such work is mostly motivated by trying to understand how individual open source projects grow and evolve.

The work presented in this paper, in contrast, analyzes the overall growth of open source, aggregating data from more than 5000 active and popular open source projects to determine the total growth of source code and number of projects. Assuming a positive correlation between work spent on open source, its total growth in terms of code and number of projects, and the revenue generated from it, understanding the overall growth of open source will give us a better indication of how significant a role open source will play in the future.

Understanding overall open source growth helps more easily answer questions about, for example, future product structures (how much code of an application is likely to be open source code?), labor economics (how much and which open source skills does a company need?), and revenue (what percentage of the software market’s revenue will come from open source?).

The work presented in this paper shows that the total amount of open source code and the total number of projects is growing exponentially. Assuming a base of 0.7% of the market’s revenue, exponential growth is a strong indicator that open

source will be of significantly increasing commercial importance. The remainder of this paper discusses our study and validates the hypothesis of exponential growth of open source.

3 Data Source and Approach

On SourceForge, the dominant open source project hosting service, there are more than 150,000 projects registered, most of which are considered inactive [1] [17]. Daf-fara estimates that as of today there are only about 18,000 active open source projects in the world [3].

For our analysis, we use the database of the open source analytics firm Ohloh.net, which has been crawling open source software code repositories since 2005 [11]. Our database snapshot contains 5122 active and popular open source projects written in 30 different programming languages covering 103 open source licenses. All data is updated on at least a weekly basis.

The database contains the most popular open source projects as measured by the number of in-links to their website. The in-links are provided by the Yahoo! search engine. The database contains data from January 1990 until May 2007. Of this time horizon, we analyze the time frame from January 1995 to December 2006. We omit data before 1995 because it is too sparse to be useful.

Ohloh.net provides high-level data like project structures and developer information, but also data that goes down to the level of individual developer actions. Specifically, Ohloh provides each individual commit action of all projects over their entire history to the extent that they are publicly available.

A commit is the action with which a developer contributes a piece of code to the project's repository. A developer's workweek typically consists of a stream of commit actions by which he or she shares the results of their work with the team, contributing to the product or project under way.

We use the amount of source code added to a project (or removed) as an approximation of the work contributed. We count code in source lines of code (SLoC), omitting empty or commented lines of code. Each commit action stored in the database lists the number of lines of code added and removed in the commit. The number of lines added or removed is calculated using the Unix diff command applied to two consecutive versions. Empty or commented lines of code are ignored. Using this data, we calculate the change in the size of a source code file by adding or subtracting the number of lines of code added to or removed from its existing size.

This data collection method gracefully handles file and directory renaming. Such renaming is modeled as if the file or directory was removed and then re-added under a new name. Both code added and code removed will have equal (large) values, so the net change is zero. This avoids any undue bias in the analysis.

Libraries are typically used across many projects. For instance, the GIMP project and the GNOME project have many libraries in common. If the lines of code for both projects were added up independently we would be double-counting the libraries, leading to skewed results. We make sure that we are not double-counting code by considering each change to the original library.

However, we cannot unambiguously identify situations where a developer adds redundant source code to the code base. Copy and paste is a common practice in software development, independently of whether it is internal, external, planned or opportunistic. To deal with this issue, we adopt two approaches.

1. In the first approach we ignore the copy and paste problem and analyze the source lines of code added. The argument is that copy and paste is a reality of software development and that the copied code is part of the project. Hence, copy and paste simply needs to be accepted.
2. In the second approach we find the average and the standard deviation for the code added over time. We ignore all commits where lines of code added is greater than average code added per commit plus three times the standard deviation. The heuristic's assumption is that by not considering such large commits we ignore all commits based on copy and paste.

An analysis of average code contribution size in commits provides a cut-off value of 3060 SLoC that we use for the heuristic. This second approach is conservative in that we ignore not only copy and paste but also commits containing new code added. So we err on the lower side of total open source contributions.

We employ these two approaches to get an upper and a lower bound for the growth in source lines of code and number of projects. We can therefore say that properties like the exponential growth observed in both the upper and lower bound curve apply to the real curve as well.

4 Analysis and Results

We first analyze growth rate and total growth in open source software code and then analyze growth rate and total growth in open source software projects.

4.1 Growth in source code

Figures 1 and 2 show plots that represent the growth in source lines of code added using Approach 1 and 2 respectively. The Y-axis shows the number of lines of code added each month and the X-axis shows the time. Each data point on the plot represents the total number of lines of code added during that month. The time frame is 1995 through 2006 for all projects. We can see an upward trend in the amount of code added over time. Both Approach 1 and 2 show a similar pattern of growth.

Table 1 shows models for the two plots. In both cases, the best fitting model is an exponential curve with an R-square value of about 0.9, giving us confidence in the validity of the claim that the amount of code added is growing exponentially.

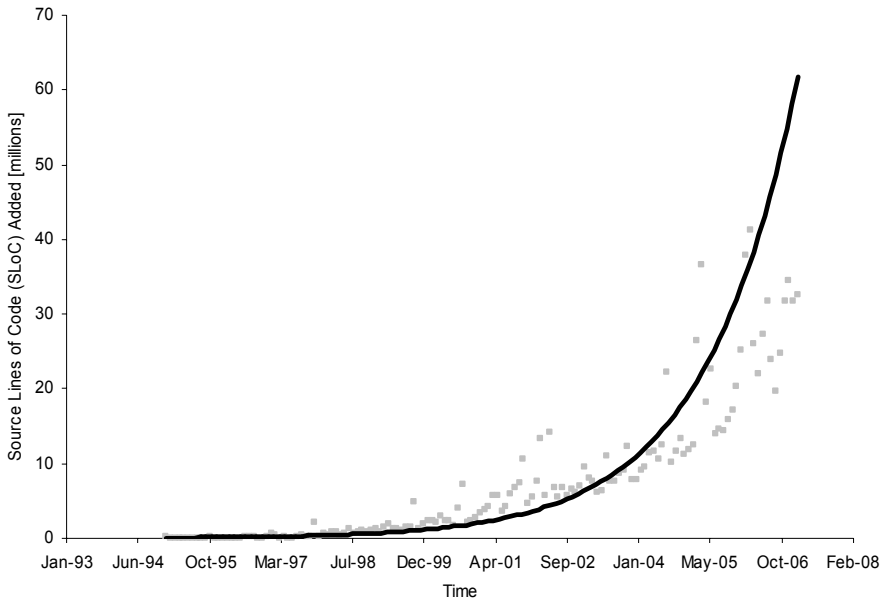


Fig. 1. Graph of source lines of code added [millions] (Approach 1)

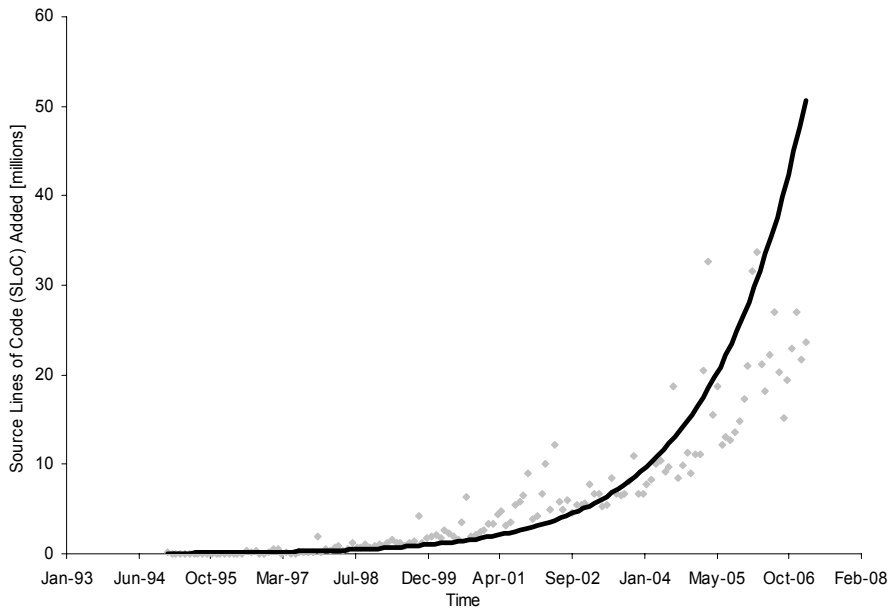


Fig. 2. Graph of source lines of code added [millions] (Approach 2)

Table 1. Model of source lines of code added

Approach	Model	R-square value
1	$y = 70833 * e^{0.0464x}$	0.901
2	$y = 64004 * e^{0.046x}$	0.897

where,
 y: Source lines of open source code added
 x: Time from Jan 1995 to Dec 2006 in months

Figure 3 shows the total number of lines of open source code over time. Table 2 shows the statistical models for the two approaches. The doubling time for Approach 1 is 12.5 months, and the doubling time for Approach 2 is 14.9 months. We observe that the total code in Approach 2 is lower than in Approach 1 but follows a similar trend. This behavior is expected as we eliminated all large commits in the second approach to exclude copy and paste contributions.

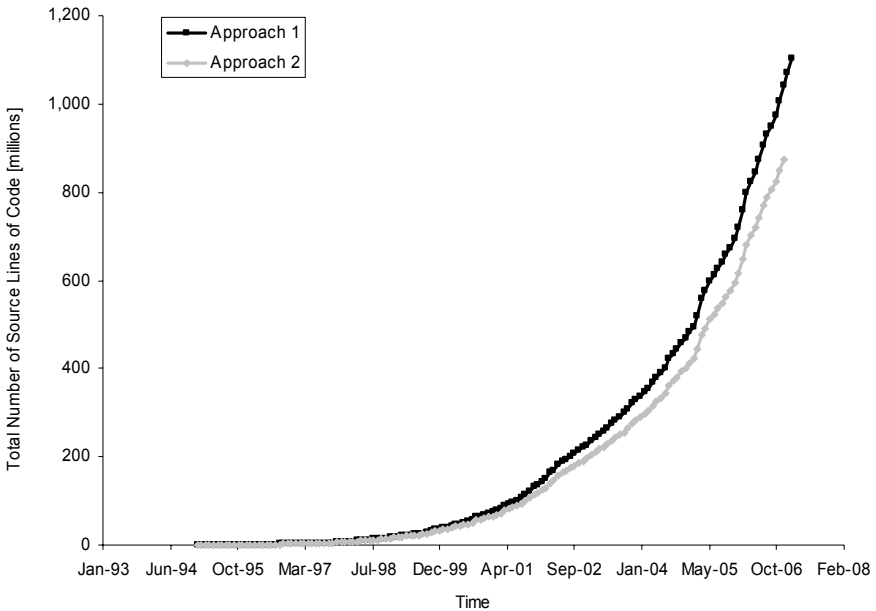


Fig. 3. Graph of total source lines of code [millions] (both approaches)

Table 2. Model of total source lines of code

Approach	Model	R-square value
1	$y = 784098 * e^{0.0555x}$	0.961
2	$y = 2E+06 * e^{0.0464x}$	0.964

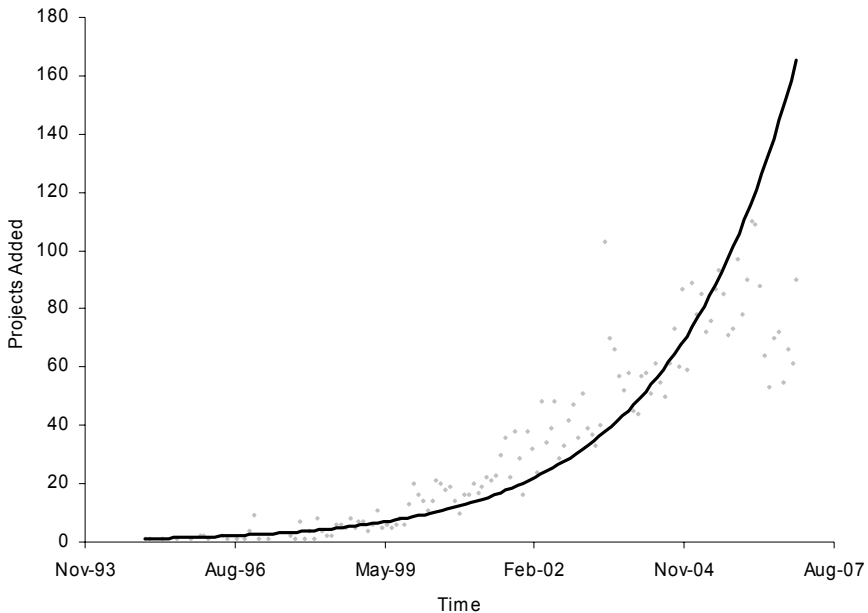
where,

y: Total open source lines of code

x: Time from Jan 1995 to Dec 2006 in months

4.2 Growth in projects

Figure 4 shows the number of projects added over time and Table 3 shows the model and its fit with the data. For each project, there is a first occurrence of a project action (for example, the initial commit action), and that point of time is considered the birth date of the project. This is the point of time when the project is counted as added to the overall set of projects.

**Fig. 4.** Graph of number of open source projects added

Large distributions like Debian are counted as one project. Popular projects such as GNU Emacs are counted as projects of their own, little known or obsolete packages such as the Zoo archive utility are ignored. Many of the projects that were included in a Debian distribution around 1998 are not popular enough today (as stand-alone projects) to be included in our copy of the Ohloh database.

And again, we get the best fit for the resulting curve for an exponential model with an R-square value of 0.88.

Table 3. Model of number of open source projects added

Model	R-square value
$y = 1.0641e^{0.035x}$	0.884
where, y: Total number of open source projects x: Time from Jan 1995 to Dec 2006 in months	

Figure 5 then shows the total number of projects and Table 4 shows the corresponding model and its fit with the data. Again, we get the best fit for an exponential model with an R-square value of 0.96. The doubling time is 13.9 months.

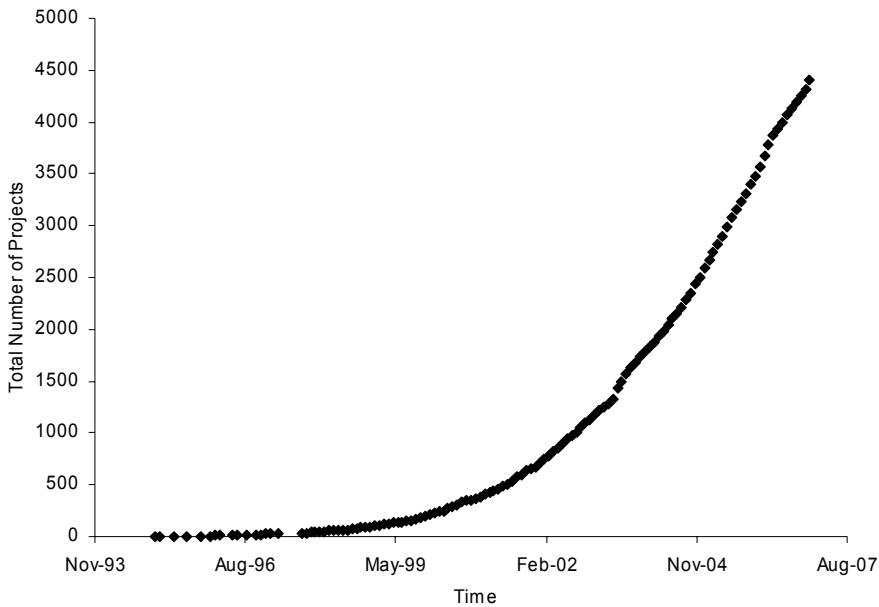


Fig. 5. Graph of total number of open source projects

Table 4. Model of total number of open source projects

Model	R-square value
$y = 7.1511e^{0.0499x}$	0.956
where, y: Total number of open source projects x: Time from Jan 1995 to Dec 2006 in months	

4.3 Review of findings

This section shows the growth of source code in open source projects as well as the growth of open source projects itself. We consistently get the best fit for the data using exponential models. The doubling time based on the exponential models is about 14 months for both the total amount of source code and the total number of projects. It should be noted that if we were to break up the data sets into separate time periods, we might find better fits for other models than the exponential model. In future work we will analyze the overall growth in distinct phases, each of which is best explained by a separate growth model.

In [13] we discuss the size and frequency of code contributions to open source projects. We can use those results to further increase our confidence in the results presented above. Specifically, the lines of code added can be assumed equal to the product of the average size of a commit in terms of source lines of code and the commit frequency. Our analysis shows that the average commit size is almost constant while the commit frequency (number of commits per week) increases exponentially between Jan 1995 to Dec 2006. This verifies our findings about the exponential growth in open source.

5 Limitations of Analysis

The quantitative analysis and the conclusions we draw have the following shortcomings and limitations.

- *Sample size.* We considered 5122 active and popular open source projects. The total number of open source projects in the world is much larger. However, Daffara estimates that of the total number only 18,000 projects (lower bound) are actually active [3]. So we believe that the sample we are using is relevant for analyzing trends and patterns in open source growth.
- *Data incompleteness.* Some amount of revision control information in open source projects has already been lost forever, as projects have moved on from no configuration management (CM) to CM with CVS and on to other CM tools, frequently dropping the history with each move. Thus, the project

history for each project is not always complete. However, for a current project, we have the most recent history, which is what is most relevant for our analysis. Thus, the lack of some of the early histories of some of the open source projects has little effect on the validity of our conclusions.

- *Project source.* A current limitation of Ohloh is that it only connects to CVS, Subversion and Git source code repositories. We believe that this limitation is not a big issue for our purposes because almost all open source projects are maintained in one of these repositories and our sample size can be considered representative.
- *Copy and paste.* Our approach to eliminating copy and paste issues (Approach 2) is limited in its effectiveness: The filter excludes a lot of good values while still allowing minor copy and paste to pass. For the purposes of our analysis, however, it is not a major issue, because we are interested in the overall trend, and even the conservative Approach 2 still validates our hypothesis of exponential growth.

We are continuing our work to iron out possible pitfalls based on these limitations. However, we believe that while the respective critiques can be made, the effects are rather limited, as argued above in each case.

6 Related Work

Several studies of the evolution of open source projects have been undertaken.

- González-Barahona *et al.* estimated the lines of code in the Debian 2.0 release and concluded that the system represents an effort of more than 14,000 person-years, which translates to about 2 billion USD [6].
- Succi *et al.* showed a linear growth rate for the GCC and Apache projects. They also showed that Linux has super linear growth [18]. They found that Linux (in 2000) violates Lehman's fourth law of software evolution.
- In contrast to this, Roy and Cordy examined the evolution of the Barcode Library and the zlib project and showed that these two smaller projects follow Lehman's laws of software evolution [16].
- Godfrey and Tu showed a super-linear increase in source lines of code over time in the Linux kernel and the VIM text editor [5].
- Robles *et al.* confirmed that the Linux kernel is growing super-linearly [15]. The NetBSD, FreeBSD, OpenBSD (until 2001) and 18 other projects showed an almost linear growth pattern.
- Koch's study of 4047 open source projects on SourceForge indicates that a quadratic growth model fits the growth of an individual project better than a linear growth model [8] [20].

- Scacchi reviews prior results on open source evolution, suggesting that the growth patterns for large open source projects are not representative for all of open source [22]. His discussion of the evolution of open source software suggests that Lehman's laws of software evolution based on closed-source systems do not apply to open source, and that further study is needed.

Most of the research listed above explores the evolution of individual projects. The growth models of projects are typically linear or quadratic. None of the related work quantitatively analyzes the total growth of open source software.

Our analysis does not focus on any particular project but on the general trend in open source software. The projects considered are independent of any particular license, language, topic or size.

7 Conclusion

The significance of open source has been continuously increasing over time. Our research validates this claim by looking at the total growth of open source. Our work shows that the additions to open source projects, the total project size (measured in source lines of code), the number of new open source projects, and the total number of open source projects are growing at an exponential rate. The total amount of source code and the total number of projects double about every 14 months.

Our results open gates for further research around the growth of open source and the acceptance of open source in industry and government. Future research should explore questions like what factors are influencing this exponential growth, how source code growth relates to the number of engaged software developers, and whether or how long open source can sustain this exponential growth.

Acknowledgments

We would like to thank Prem Devanbu and Gregorio Robles for their feedback on earlier versions of the paper as well as their encouragement for the work presented. We also would like to thank Oliver Arafat and Mario Fernandez for proofreading the paper.

References

- [1] Comino, S, Manenti, F.M., Parisi, M. L. *From Planning to Mature: On the Determinants of Open Source Take Off*. Department of Economics Working Papers 0517, Department of Economics, University of Trento, Italia. 2005.
- [2] Crowston, K. and Scozzi, B. Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development. *IEE Proceedings---Software Engineering*, vol. 149, no. 1, 2002: 3-17.

- [3] Daffara, C. How Many Stable and Active Libre Software Projects? Retrieved on Sept 13, 2007, from <http://flossmetrics.org/news/11>.
- [4] Software & Information Industry Association. *Packaged Software Industry Revenue and Growth, 2006*. Available from <http://siiia.net/software/>
- [5] Godfrey, M., Tu, M. Growth, Evolution, and Structural Change in Open Source Software. In *Proceedings of the 4th International Workshop on Principles of Software Evolution*. ACM Press, 2001: 103-106.
- [6] González-Barahona, J., Ortuño Pérez, M., de las Heras Quirós, P., Centeno González, J., Matellán Olivera, V. Counting potatoes: The Size of Debian 2.2. Retrieved on Sept 13, 2007, from <http://people.debian.org/~jgb/debian-counting/counting-potatoes/>.
- [7] Haruvy, E., Wu F. and Chakravarty S. Incentives for Developers' Contributions and Product Performance Metric in Open Source Development: An Empirical Exploration. University of Texas Working Paper.
- [8] Koch, S. Evolution of Open Source Software Systems---A Large-Scale Investigation. In *Proceedings of the 1st International Conference on Open Source Systems (OSS 2005)*.
- [9] Lawton, M., Notarfonzo, R. Worldwide Open Source Software Business Models 2007–2011 Forecast: A Preliminary View. IDC Inc.
- [10] Netcraft. Netcraft Web Server Survey. Netcraft, 2007. Retrieved on Sept 13, 2007, from <http://survey.netcraft.com/Reports/200708/byserver/>.
- [11] Ohloh Corporation. See <http://www.ohloh.net>.
- [12] Raymond, E. S. *The Cathedral and the Bazaar*. O'Reilly & Associates, 1999.
- [13] Deshpande, A. Riehle, D. Continuous Integration in Open Source Software Projects. Submitted to the *4th International Conference on Open Source Systems (OSS 2008)*.
- [14] Robles, G., Gonzalez-Barahona, J. M., Michlmayr, M., and Amor, J. J. Mining Large Software Compilations Over Time: Another Perspective of Software Evolution. In *Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR 2006)*. ACM Press, 2006: 3-9.
- [15] Robles, G., Amor, J. J., Gonzalez-Barahona, J. M., and Herraiz, I. Evolution and Growth in Large Libre Software Projects. In *Proceedings of the Eighth International Workshop on Principles of Software Evolution (IWPSE 2005)*. IEEE Computer Society, 2005: 165-174.
- [16] Roy, C. K. and Cordy, J. R. Evaluating the Evolution of Small Scale Open Source Software Systems. See <http://citeseer.ist.psu.edu/761885.html>.
- [17] SourceForge. See <http://www.sourceforge.net>.
- [18] Succi, G., Paulson, J., Eberlein, A. Preliminary Results From an Empirical Study on the Growth of Open Source and Commercial Software Products. In *EDSER-3 Workshop (2001)*: 14-15.
- [19] Walli, S., Gynn, D., Rotz, B. V. The Growth of Open Source Software in Organizations: A Report. Retrieved on Sept 13, 2007, from http://optaros.com/en/publications/white_papers_reports/the_growth_of_op_en_source_software_in_organizations.

- [20] Koch, S. Software Evolution in Open Source Projects—A Large-Scale Investigation. In *Journal of Software Maintenance and Evolution: Research and Practice* 2007; 19: 361-382.
- [21] Karim, R., Lakhani, R.G. Wolf. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. In *Perspectives on Free and Open Source Software*. MIT Press, 2005: 3-22.
- [22] Walt Scacchi. Understanding Open Source Software Evolution. In *Software Evolution and Feedback*. John Wiley & Sons, 2006.

Adoption of Open Source in the Software Industry

Øyvind Hauge, Carl-Fredrik Sørensen, and Reidar Conradi
Norwegian University of Science and Technology
{oyvind.hauge|carl.fredrik.sorensen|reidar.conradi}@idi.ntnu.no

Abstract. Is Open Source Software (OSS) undergoing a transformation to a more commercially viable form? We have performed a survey to investigate the adoption of OSS in the Norwegian software industry. The survey was based on an extensive screening of software companies, with more than 700 responses. The survey results support the transformation predicted by Fitzgerald [4]. Close to 50% of the software industry integrate OSS components into vertical solutions serving all major business sectors. In addition, more than 30% of the 95 respondents in our survey have more than 40% of their income from OSS related services or software. The extensive adoption of OSS in the software industry may be a precursor of the OSS adoption in other business sectors.

1 Introduction

Open source software (OSS) is predicted to transform “into a more mainstream and commercially viable form” [4], where companies play an increasingly more important role. A premise for this transformation is increased commercial participation in the development of OSS products and increased use of OSS in vertical domains. However, only a few surveys provide empirical findings which support this assumed transformation, and most of these focus on the use of desktop tools and horizontal infrastructures like the LAMP stack. Is really OSS undergoing a transformation?

To answer this question we have performed a large scale survey in the Norwegian software industry. Our analysis shows that close to 50% of the software industry integrate OSS components into vertical solutions targeting customers from all major business sectors. In addition, more than 30% of the respondents in our survey have more than 40% of their income from OSS related services or software.

Our results show that the adoption of OSS in the Norwegian software industry is significant. The industry’s contribution to the OSS community is however limited. Nevertheless, it is reason to believe that OSS is actually undergoing a transformation into a more commercially viable form. The use of OSS in the software industry may eventually influence the rest of the market when software companies integrate OSS into their products. However, a lack of software companies adopting OSS may hamper the adoption of OSS in other sectors [15].

Please use the following format when citing this chapter:

Hauge, Ø., Sørensen, C.-F. and Conradi, R., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 211–221.

2 Related Work

Estimating the market share of OSS is a comprehensive task. Nevertheless, several attempts have been made e.g. [5, 6, and 18]. Common to most of these is their focus on a few products like the LAMP stack and end-user applications like mail or office tools. One example is the Netcraft¹ survey of web servers on the Internet. While several consultancy companies have attempted to estimate the adoption of OSS, we rather focus on research published through academic channels.

Without providing any numbers, Glynn et al. conclude that OSS has had significant penetration in the software/consultancy and service/communication sector, but that it is more limited in the government/public sector [7]. Studies from the UK [17], Finland [15], and the U.S. [16] report only limited OSS adoption in the public sector with Linux as the only exception. Linux was used by more than 50% of the respondents in both the study from Finland and the U.S. Together with the other elements of the LAMP stack, Linux is quite frequently used in other sectors as well [6]. However, this adoption varies from country to country, on company size, and between sectors. For a mixed sample from industry and public sector, the use or planned use is reported to be as low as 17.7% in Sweden and as high as 43.7% in Germany [6]. Furthermore, numbers vary between about 10% and 75% for different strata [6]. A survey on Australia's top companies reports that 26% of the respondents used a varied spectrum of OSS products [8]. With the exception of Linux, Apache HTTP Server and perhaps a few others, most surveys report that less than 30% of the respondents have adopted OSS. Yet, little is known about the extent of the internal adoption of OSS in these companies.

In studies focusing on the software sector, 44% of the companies in a Finnish sample use OSS in their business [13] and in a study on Off-the-Shelf development, 44 or 38.3% of the 115 projects use OSS components [11]. Without being able to provide any numbers representative for the whole population, an Italian study found that software companies using OSS commonly adapt or build on top of these OSS products [1].

The transformation predicted by Fitzgerald involves company participation in the development of OSS products [4]. Companies are already known to be contributing by allowing employees spend their time at work participating in OSS projects [10]. Companies are among others involved in 97 of the 300 most active SourceForge projects [2]. However, this is most likely not representative for all of SourceForge's more than 170 000 projects. A Swedish survey also found that several companies actively contribute to OSS projects [12].

We see that companies and organizations have adopted OSS and that they are involved in the development of OSS. There are however only a limited number of empirical findings which show the extent of this adoption and the demography of these companies. This paper will provide results which quantifies the adoption of OSS components in the Norwegian software industry.

¹ <http://news.netcraft.com/>

3 Survey Method

The purpose of the study was to investigate to what extent the Norwegian software industry approaches OSS development. As an expansion of [9], we carried out a nationwide survey to investigate this matter.

3.1 Population: The Norwegian Software Industry

Legal entities in Norway are registered in The Norwegian Central Coordinating Register for Legal Entities² (CCRLE) with a Nomenclature Generale des Activites Economiques dans L'Union Europee (NACE) code. Based on 2005 data from CCRLE and other registers, Statistics Norway³ (SSB) reports that about 70 000 employees, or 4.7 % of all employees in Norway, are employed in the whole ICT sector [14]. In addition, the sector has a turnover of about €22 billion [14].

Table 1. The Norwegian 72.xx sector based on data from CCRLE 2007

Sub sector	NACE	Entities
Computer and related activities	72.00	26105
Hardware consultancy	72.10	251
Software consultancy and supply	72.20	21559
- Publishing of software (software houses: resale)	72.21	1295
- Other software consultancy and supply (single sale)	72.22	20264
Data processing	72.30	489
Database activities	72.40	2916
Maintenance & repair: office, accounting and comp. machinery	72.50	733
Other computer related activities	72.60	163

The ICT sector in Norway includes telecommunication (64.20), ICT manufacture industry (32.xx), ICT wholesale and retail trade (51.8x), and the soft- and hardware sector (72.xx). Based on CCRLE data from 2007, we found that approximately 26 000 legal entities and 38 500 employees constitute the soft- and hardware sector, see Table 1. This gives an average company size of about 1.5 employees. According to SSB only about 13 000 of these legal entities are active companies. More than 70% of these have less than one full time employee and about 1300 have five or more employees [14]. We will in this paper focus on the software sector (72.2x).

² <http://www.brreg.no/>

³ The Norwegian counterpart to the U.S. Census Bureau <http://www.ssb.no/>

3.2 The Sampling Process

Data from CCRLE helped us constructing a close-to representative sample of software companies [3], with a focus on software (72.21) and consultancy (72.22) companies with more than five employees. However, the sample also included companies from the other 72.xx sub-sectors and companies with fewer than five employees. The purpose of the sampling process illustrated in Fig. 1 was twofold. First, estimate the share of companies integrating OSS components into their products. Second, create a sample for our survey consisting of companies using OSS components.

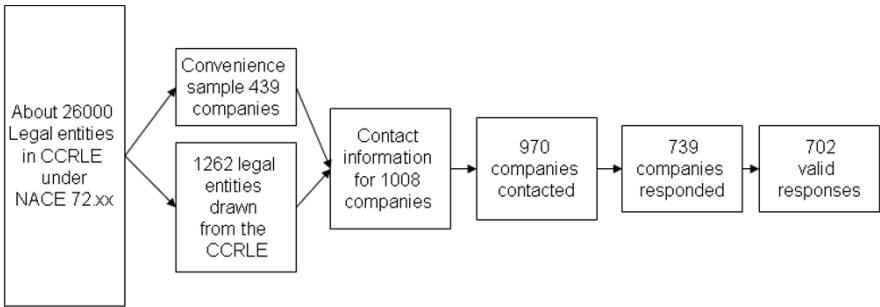


Fig. 1. The sampling process

Step 1: The sample was constructed based on a convenience sample of 439 companies and a stratified random sample of 1262 legal entities from CCRLE. The convenience sample was based on stratified random samples two from earlier studies and supplemented with companies from our knowledge and companies appearing in the media. The strata were defined according to the business organization form, the 72.xx sub-sectors, and the number of employees.

Step 2: Then, the two lists were merged. 300 duplicate entries were removed during this merger. Several companies occurred in both samples and some companies were registered with more than one legal entity, typically larger companies. We used data from CCRLE and the Internet to find web-sites and email addresses for the companies. Another 395 or 31.3 % of the 1262 randomly selected legal entities were removed from the list because no contact information could be found. Knowing that only about 50% of the companies in the sector were active, this was not a surprise. The vast majority of these companies were small and most likely inactive companies. The final list contained contact information for 1008 companies from the Norwegian software industry.

Step 3: The screening process was carried out by sending the companies a brief request on email containing the questions stated below. About 200 of the companies from the convenience sample were contacted in March 2007 and the rest in June/July. One reminder was sent by email in September. The 200 companies

contacted in March were only asked the first three questions while the remaining companies were asked all four questions.

1. How many employees do you have in Norway?
2. Are you doing software development in Norway?
3. Do you use open source components in your products or services (other than Linux, Apache HTTP Server, Eclipse, PHP/Perl, MySQL etc.)?
4. Do you participate in or run any open source projects?

38 of the 1008 email addresses did not work, leaving 970 companies. 201 of these companies came from the convenience sample, 555 from the stratified random sample, and 236 companies were included in both samples. 739 companies replied which give us a response rate of 73.3%. 32 companies responded that their company was inactive or about to be dissolved, one company did not want to participate, and another four duplicate legal entities were found, leaving 702 or 69.6% valid responses. The response rates were similar across most of the different strata (size and sector). The names of the respondents and their email addresses were stored together with the other contact information.

Step 4: 569 or 81.1% of the 702 companies in our screening process confirmed that they perform software development. These companies make the basis for further analysis. The percentage of companies involved in software development is similar across different size and business types. However, when looking at sectors, the percentage varies from 73.6% (72.40 Database activities) to 90.2% (72.20 Publishing of software).

3.3 The Survey Process

Close to 50% of the 569 companies constituting our sample integrate OSS components into their products. 204 of these companies were invited to participate in a web survey. The survey contained three parts focusing on (1) development of a commercial OSS product, (2) integration of OSS components into a software product, and (3) demographic information. The respondents should answer based on their experiences with the development of a typical software product containing OSS components. This product was selected by respondents and we had no control over this selection. To learn more about the companies and to increase the response rates, every second company ordered by size was contacted by phone. The companies were asked if they could participate and were sent an email with instructions if they accepted our invitation. The other half was invited to participate through email. One reminder was sent by email about a month later. 12 or 5.9% of the 204 companies could not or did not want to participate. Nevertheless, 95 of the 204 companies or 46.6% completed the survey. Of these 95, 21 were only involved in software development without directly developing software products, for instance consultancy companies providing developers to external customers. This left 74 or 36.3% valid responses for the two main parts.

4 Results

This section presents results from both the screening process and from the survey.

4.1 Selection and Integration of OSS Components

Out of the 569 companies constituting our sample, 266 or 46.9% integrate OSS components into their software solutions. This use goes beyond merely using OSS operating systems, databases, infrastructure, development tools, and programming languages. The companies actually find, evaluate, and integrate OSS components into their software solutions. The integration of OSS components happens less frequently in software houses. Only 34.1% of the companies registered in sector 72.21, use OSS components in their products, see Table 2.

Table 2. Adoption of OSS components distributed over sectors

Sector	Sample Size	OSS Adoption
72.21 Publishing of software	129	34.1%
72.22 Other software consultancy and supply	328	51.5%
72.30 Data processing	18	38.9%
72.40 Database activities	39	53.8%
Other	55	47.3%

From Table 3 we see that large companies integrate OSS components more often into their products than smaller companies. 56.9% of the companies with more than 100 employees use OSS and 50.0% of the companies with 25 to 99 employees integrate OSS components into their products compared to around 43% of the companies with between 2 and 24 employees. Companies with one or less than one full time employee, use OSS components somewhat more frequently.

Table 3. Adoption of OSS components distributed over the number of employees

Number of employees	Sample Size	OSS Adoption
0 to 1	33	48.5%
2 to 4	61	42.6%
5 to 9	80	43.8%
10 to 24	189	43.9%
25 to 99	146	50.0%
More than 100	58	56.9%

66 companies completed the second part of the survey based on their experiences from the development of a software product containing OSS component. The products delivered by these companies serve all main business sectors with a small

emphasis on the public and health sector. The functionality of these products was directed mainly towards web/portals and enterprise solutions. The respondents classified 40.9% of the products as domain specific and 36.4% as differentiating end-user products.

Even though OSS components can reduce the development effort substantially, they are in most cases integrated as part of a larger solution. In 72.7% of the products, OSS components provide less than 40% of the functionality of the end product. The number of OSS components is also kept low. 68.2% of the products contain less than six OSS components and 83.3% contain less than eleven OSS components. The effort spent developing these products during the last year, range from less than one (1) person-month to between 101 and 500 person-months.

4.2 OSS Related Activities

During the last year, 75.8% of the companies have developed between one and three software products containing OSS components. In one extreme case, one company had developed more than 50 products containing OSS components. In Part 3 of the survey, we requested the respondent to estimate how much of the company's income is generated by OSS related services or software development, see Table 4. Even though 41 of the 95 respondents answered less than 20% and 22 answered "don't know", 29 or 30.5% answered that more than 40% of their income comes from OSS related services or software.

Table 4. Income from OSS related services and software development

Income from OSS	Number of companies
NA/Don't know	22
0%	8
1-20%	33
21-40%	3
41-60%	7
61-80%	9
81-99%	6
100 %	7

4.3 Participation in OSS Projects

In total 368 of the 569 companies developing software responded to the fourth screening question. 60 or 16.3% of the respondents said that employees in their company participated in OSS projects. This participation was in some cases part of their job and in other more a hobby. Another 18 or 4.9% of the 368 companies said they have their own OSS project. However, through the researchers' previous

experience with some of these companies we would say that the OSS projects are only an important part of the business for a few of them.

30 of the 66 companies completing Part 2 of the survey answered that they interacted with or participated in OSS projects during the development of their product. This interaction and participation was in all but three cases not organized through the company but rather left up to the individual developer.

5 Discussion

The use of OSS in Norwegian software industry is significant. Close to 50% of the companies developing software have integrated OSS components into one or more of their products and more than 30% of the respondents to our survey get over 40% of their income from OSS related services or software. The use of OSS in software houses (72.21) living off the sales of software licenses is somewhat lower than in other software sectors. This could be caused by reciprocal OSS licenses (e.g. GPL) which requires derivative products to be released under the same license, thus removing the software houses profits from sales of licenses. Another conceivable explanation is that companies focusing on development of their own software products are involved in the development of fewer products per year than consulting companies serving several different customers.

The products developed by the respondents served all major business sectors and 77.3% of them were classified as domain specific or differentiating end-user products. Thus, we can conclude that OSS is used in vertical products targeting all business sectors.

We observed increasing OSS use in relation to increasing company size. However, Lundell et al. observed that companies with more than 250 employees seemed more conservative towards OSS adoption [12], Bonaccorsi et al. found that size does not favor OSS adoption [1], and Ghosh et al. found variations in the adoption of OSS over company size, countries, and sectors [6]. The relation between size and OSS adoption needs to be investigated in future research. However, we see two possible explanations for this increased use of OSS. First, small companies commonly focus on a limited number of customers and specialize on a small set of different technologies. Several such companies replied that they were not using OSS because they focused only on one not-OSS-compatible technology. While large companies, often serve many customers using several different technologies, including OSS. Second, large companies hire more people. Because they hire more people it is more likely that they employ people with prior experiences with OSS.

Comparing the results from this survey with other results is a bit complicated. First, the number of related studies is fairly limited. Second, while we focused on integration of OSS components, most studies include all kinds of OSS products. Third, other studies focus on other sectors than the software sector. The 46.9% adoption of OSS components in the software sector is therefore lower than some of the extreme results in [5, 6]. If our survey had included all kinds of OSS, we suspect

the percent-age of OSS users to be significantly higher. On the other hand, our results are in line with the results from [13] where 44% of the software companies used OSS in their business.

Generalization to other countries is made somewhat more difficult because of the variations found in other studies [6]. There are also factors which influence the adoption of OSS in the Norwegian software sector. While the effects of these factors must be further examined, we believe the size of the companies, influence the OSS adoption. Most software companies in Norway are small or medium sized. Many have relatively few and mostly domestic customers, and they have also chosen to focus on a limited set of technologies. In addition, the Norwegian government has the last years increased its focus on open standards and OSS through public reports and the establishment of a national centre of expertise of OSS. Furthermore, there seems to be a push in Norway towards increased use of agile methods. While using such methods it is important to get something up and running as fast as possible. This and the fact that the cost of personnel in Norway is quite high may encourage higher re-use of code and components, including OSS.

Industrial participation in OSS projects seems limited and managed on an individual level. Only 16% of the software companies confirm that they do participate in the development of one or more external OSS products. This number has however some uncertainty. First, 200 of the companies in the screening process were not asked whether they participated in any OSS projects. Second, there is some confusion about what participation is. Some companies answered “we do not participate but we report bugs and share occasional bug fixes” while others answered “we do participate with some bug reports and bug fixes”. However, we interpreted both these statements as participation. Third, participation in OSS projects is in most cases managed on a personal level. Knowing what all other employees are doing is difficult if not impossible for the respondents in our screening process and the number of companies participating in OSS projects could therefore be higher.

The sample has an intentional bias towards companies with more than five employees. This bias was reinforced by the fact that we were unable to find contact information for several small and probably inactive companies. The majority of companies without a web site are most likely inactive since nowadays the Web is considered the most important communication channel. To aid the sampling we benefited from CCRLE and the NACE sector codes. While this classification was of great help, the software industry is an industry with rapid and frequent changes. As a consequence of these changes, central registers such as CCREL are not up-to-date at all times. For example, only about 90% of the companies under 72.20 “Publishing of software” actually develop software.

Response rates of 73.3% for the screening process and 36.3% valid responses in the main survey is decent compared to many other studies but low response rates is one of the challenges with survey research. Even though there is room for improvement, we have been able to get responses from a large and close to representative sample of the Norwegian software industry. The research design is

well documented and replicating the survey in another setting should be easy, though labor-intensive.

6 Conclusion

Results from our study show limited company involvement in the development of OSS products but widespread use of OSS components. By integrating OSS into vertical products serving all major business sectors, the software industry will contribute to wider adoption of OSS. The software industry has clearly started to adopt OSS products and contribute to the transformation of OSS into a commercially viable form. However, this transformation is far from completed.

The results presented here are currently followed up in several ways. In parallel to this survey, we have also approached OSS adoption through qualitative studies. Data from both the survey and these qualitative studies is currently being analyzed. Findings from these analyses will provide a basis for further research. This survey focuses on the Norwegian software sector, which is dominated by small and medium sized companies. The adoption of OSS may be different in other sectors and in other countries. We are therefore looking at the possibilities of conducting similar surveys in both other sectors and European countries. To understand the trend of OSS adoption, we are also considering a replication of the survey in Norway. Furthermore, the survey identified a few companies developing their own OSS products. We plan to follow up on these companies to try to understand if they manage to attract and sustain communities and how they interact with these communities.

References

1. Andrea Bonaccorsi, Silvia Giannangeli, and Cristina Rossi. Entry Strategies under Competing Standards: Hybrid Business Models in the Open Source Software Industry. *Management Science*, 52(7):1085-1098, July 2006.
2. Andrea Bonaccorsi, Dario Lorenzi, Monica Merito, and Cristina Rossi. Business Firms' Engagement in Community Projects. Empirical Evidence and Further Developments of the Research. In *Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development FLOSS'07*, page 13, Minneapolis, US, 2007. IEEE Computer Society.
3. Reidar Conradi, Jingyue Li, Odd Petter N. Slyngstad, Vigdis By Kampenes, Christian Bunse, Maurizio Morisio, and Marco Torchiano. Reflections on Conducting an International Survey of Software Engineering. In June Verner and Guilherme H. Travassos, editors, *Proceedings on International Symposium on Empirical Software Engineering ISESE'05*, pages 214-223, Brisbane, Australia, 2005.
4. Brian Fitzgerald. The Transformation of Open Source Software. *MIS Quarterly*, 30(3), 2006.

5. Rishab Aiyer Ghosh. Study on the Economic Impact of Open Source Software on Innovation and the Competitiveness of the Information and Communication Technologies (ICT) Sector in the EU. Technical report, UNU-MERIT, 2006.
6. Rishab Aiyer Ghosh, Gregorio Robles, and Ruediger Glott. Free Libre and Open Source Software: Survey and Study. Technical report, International Institute of Infonomics, University of Maastricht, 2002.
7. Eugene Glynn, Brian Fitzgerald, and Chris Exton. Commercial Adoption of Open Source Software: An Empirical Study. In Proceedings of International Conference on Empirical Software Engineering, pages 225-234, Noosa Heads, Australia, 2005.
8. Sigi Goode. Something for Nothing: Management Rejection of Open Source Software in Australia's Top Firms. *Information & Management*, 42(5):669-681, 2005.
9. Øyvind Hauge, Carl-Fredrik Sørensen, and Andreas Røsdal. Surveying Industrial Roles in Open Source Software Development. In Joseph Feller, Brian Fitzgerald, Walt Scacchi, and Alberto Sillitti, editors, Proceedings of the Third International Conference on Open Source Systems, pages 259-264, Limerick, Ireland, 2007. Springer.
10. Karim R. Lakhani and Robert G. Wolf. Why Hackers Do What They Do: Understanding Motivations and Effort in Free/Open Source Software Projects. In Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani, editors, Perspectives on Free and Open Source Software, pages 3-23. MIT Press, 2005.
11. Jingyue Li, Reidar Conradi, Odd Petter N. Slyngstad, Christian Bunse, Umair Khan, Marco Torchiano, and Maurizio Morisio. An Empirical Study on Off-the-Shelf Component Usage in Industrial Projects. In Frank Bomarius and Seija Komi-Sirvio, editors, Proceedings of the 6th International Conference on Product Focused Software Process Improvement PROFES'2005, pages 54-68. Springer, 2005.
12. Björn Lundell, Brian Lings, and Edvin Lindqvist. Perceptions and Uptake of Open Source in Swedish Organisations. In Ernesto Damiani, Brian Fitzgerald, Walt Scacchi, Marco Scotto, and Giancarlo Succi, editors, Proceedings of The Second International Conference on Open Source Systems, pages 155-163, Como, Italy, 2006. Springer.
13. Uolevi Nikula and Sami Jantunen. Quantifying the Interest in Open Source System: Case South-East Finland. In Marco Scotto and Giancarlo Succi, editors, OSS 2005: Proceedings of the First International Conference on Open Source Systems, 11-15 July 2005, Genova, Italy, pages 192-195, 2005.
14. SSB. StatBank Norway, 2007. <http://statbank.ssb.no/statistikkbanken/>, accessed 2007-08-01.
15. Mikko Välimäki, Ville Oksanen, and Juha Laine. An Empirical Look at the Problems of Open Source Adoption in Finnish Municipalities. In Proceedings of the 7th International Conference on Electronic Commerce ICEC'05, pages 514-520, Xi'an, China, 2005. ACM.
16. Shahron van Rooij. Open Source software in US higher education: Reality or illusion? *Education and Information Technologies*, 12(4):191-209, December 2007.
17. Teresa Waring and Philip Maddocks. Open Source Software implementation in the UK public sector: Evidence from the field and implications for the future. *International Journal of Information Management*, 25(5):411-428, October 2005.
18. David A. Wheeler. Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!. http://www.dwheeler.com/oss_fs_why.html, accessed 2007-12-09.

Migration Discourse Structures: Escaping Microsoft's Desktop Path

Leonhard Dobusch
Free University Berlin – School for Business & Economics,
Garystr. 21, 14195 Berlin,
leonhard.dobusch@wiwiss.fu-berlin.de
WWW home page: <http://www.pfadkolleg.de>

Abstract. Most studies of FOSS organizational migration projects focus solely on technological and economical aspects, neglecting the importance of organizational discourse structures for migration decisions as well as success. In looking at the case of the municipality of Munich this paper uses structuration theory in combination with discourse analysis to explain why and how in this case actors were able to overcome strong barriers to migration in the field of desktop software.

Keywords: Open Source Software; Path Dependency; Technology Adoption; Structuration Theory; Discourse Analysis

1 Introduction

Of all recent examples of organizations switching their software environment to Free/Open Source Software (FOSS) the municipality of Munich in Germany received significant attention, at least in the media. The reasons for this are the following:

- Munich decided to migrate the whole desktop software environment including operating system and office suite to FOSS – not only server software or office suits as had done many organizations before.
- With approximately 14.000 desktops the municipality in Munich is by far larger than other municipal predecessors such as for example Schwäbisch-Hall with about 225 desktop PCs [19].
- Only few weeks before the migration decision in May 2003 Microsoft's CEO Steven Ballmer met with Munich's mayor Ude to convince him not to do so and failed.

But aside these aspects the migration decision and process in Munich is also a critical case [34] for studying why and how organizations take the lead in adopting a minority system in markets with strong network effects, often referred to as “path

Please use the following format when citing this chapter:

Dobusch, L., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succì; (Boston: Springer), pp. 223–235.

dependent” [6, 3]¹. Whereas many studies of FOSS adoption, especially in the domain of information systems, focus on technical implementation and migration issues (e.g. [22]) and do not distinguish between desktop and server usage (e.g. [1]), the organizational processes associated with these migrations are often overlooked or underrated.

In combining structuration theory [12, 20, 21, 24] with concepts of discourse analysis [14, 4, 5] this paper tries to conceptualize why and how organizations take the lead in escaping Microsoft’s desktop software path.

2 FOSS Adoption as a Case of Path Breaking

2.1 Path dependency in software markets

Following Shapiro and Varian [23] as well as Varian et al. [29] there are several mechanisms in software markets such as network effects, investment and learning spirals or complementarity (see Table 1) that differentiate them from “classic markets”. All these mechanisms are recursively self-reinforcing and thus increase switching costs of individual adopters over time, making a migration to FOSS an expensive and laborious task. Following David’s [6, 7] notion of path dependency, Microsoft’s monopoly position in the market for desktop operating systems and office suites doesn’t prevail because of the superiority of its products but because of their customer’s lock-in on them.

Table 1. Selection of mechanisms leading to switching costs in desktop software markets

Mechanism	Example
Direct and indirect network effects	The larger the installed base of a piece of software, the greater the benefits (also for the individual actor) of adopting it. [10]
Investment and learning spirals	Investments into a specific piece of software lead to even more investments into the same, creating a growing stock of idiosyncratic assets. [33]
Complementarity	Two (or more) in principle autonomous mechanisms – e.g. software diversity and organizational decentralization – reinforce one another. [26]

However, the logic that leads to path dependency of (in particular: organizational) actors in software markets results from mutual and recursive interdependencies between technological and social structures. Orlikowski [20, 21] calls this the “duality of technology”:

¹ Shapiro and Varian [23] even call the market for desktop software „everyone’s favourite example” for concepts like “lock-in” or “increasing returns”.

The duality of technology identifies prior views of technology – as either objective force or as socially constructed product – as a false dichotomy. Technology is the product of human action, while it also assumes structural properties. That is, technology is physically constructed by actors working in a given social context, and technology is socially constructed by actors through the different meanings they attach to it and the various features they emphasize and use. [20:496]

In an organizational context the mechanisms described in Table 1 do not effect software adoption directly but are mediated through intra-organizational dynamics and structures. Stones' [24] "quadripartite" notion of structuration [12] was originally developed for the individual actor but seems to be applicable on the organizational level, as well (see Figure 1). The main point of structuration theory is to acknowledge the "duality at work in which agents and structures are not kept apart but in which they are mutually constitutive of one another" [24:21].

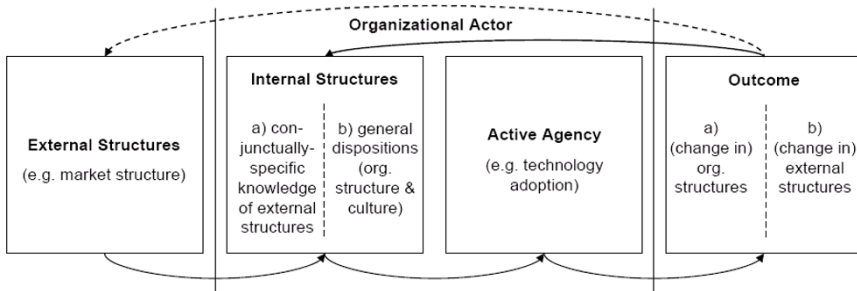


Fig. 1. Structuration model of technology adoption

2.2 Path breaking FOSS?

The reasons why there is still competition in the market for desktop PC operating systems lie in the idiosyncratic characteristics of recently developed FOSS alternatives such as GNU/Linux which differ in development process [31], license fees, access to source code, and strong competition among several suppliers in a way that is impossible to imitate for Microsoft if it wants to prolong its business model.² Nevertheless, the barriers to migration especially in the market for desktop operating systems – contrary to the server market – still seem to prevent Microsoft from serious competition in this field [13]. Especially large organizations hesitate to

² For an overview of the differences between Free/Open Source and proprietary software, see the anthology by [11].

switch their desktop software environment to alternatives provided by open source software competitors ([32], see Table 2), and still little is known why and how they do so.

Table 2. FOSS desktop usage in organizations in three countries (taken from Wichmann [32])

Size*	UK	Sweden	Germany
Small	7,6%	3,4%	13,7%
Large	2,0%	3,2%	6,5%

* < 500 employees = small; > 500 employees = large

What we do know, however, is that technological and economic reasons may play a decisive role in the question whether to adopt FOSS or stay on the Microsoft path, as Varian and Shapiro ([30:12]; italics by L.D.) summarize their review of TCO comparisons in the literature:

There have been several attempts to compare the TCO of Windows and of Linux in various computing environments. In most of the studies the difference in TCO is on the order of 10 or 15 percent. This difference is not large; a 10 percent difference in TCO could easily be swamped by local conditions, random events, and other considerations. To a first approximation, it seems reasonable to suppose that neither of these two platforms has a striking advantage over the other in terms of conventional measures of TCO.

So the question remains why and how do organizations then decide to migrate their desktop software environment to FOSS alternatives in spite of strong network effects? This paper suggests taking a look at the organizational migration discourse.

3 Method Section

3.1 Data collection

Following Yin [34:8], the unique strength of a case study is “its ability to deal with a full variety of evidence”, including documents, artifacts, interviews, and observations. To live up to this potential and for triangulation reasons, over a two year period from 2006 to 2007 data was collected in form of open-ended interviews with actors on different organisational levels, transcripts of talks given at practitioner conferences, archival documents including the “Client Study” of Unilog Integrata (consulting firm, [28]) and media coverage. All data was integrated into a case study database (see Table 3) but only transcripts were included in the discourse analysis.

Table 3. Case Study database

transcripts	
interviews / persons	7 / 7
talks / persons	3 / 3
Σ transcripts / persons	10 / 9
per functional area (transcripts / persons):	
political administration	2 / 2
central IT	5 / 3
decentral IT	2 / 2
external*	2 / 2
media coverage (2001-2007)**	
articles (online / print)	68 / 34
archival documents	
sets of slides	11
agenda papers	31
miscellaneous	7
* "external" includes service provider and consultants	
** articles from the following sources from 2002-2007 : Heise.de, Computerwoche, Frankfurter Allgemeine Zeitung, Frankfurter Rundschau, Süddeutsche Zeitung	

3.2 Discourse analysis

In adopting the cyclical ideal of qualitative research [25], the data analysis was divided into three parts that were not undertaken in strict consecutiveness: Inductive generation of theoretical categories of both mechanisms enforcing and weakening path dependency is complemented with chronological process descriptions. These two parts are connected in a final theoretical integration, using archival documents and media coverage to cross-check interview data and for the right temporal ordering.

For categorization, the literally transcribed interviews and talks were reduced to 692 paraphrases as the basis for coding the data. Then, in multiple, consecutive rounds a system of coding categories was developed with focus on the meta-category "barriers and drivers for the adoption of an alternative (i.e. non-Microsoft) desktop operating system" [16, 18]. These inductively generated categories were then classified as predominantly covering "Framing" or "Program" aspects of the migration discourse – a dichotomy taken from Campbell's [4, 5:94] typology of ideas, which differentiates between the two as follows:

- Programs are "ideas as elite prescriptions that enable [...] the charting of a clear and specific course of action"
- Frames are "ideas as symbols and concepts that enable decision makers to legitimise programs to their constituents".

In integrating these categories in an organizational structuration model as it is presented in Figure 2, the translation of discourse into organizational technology adoption can be conceptualized.

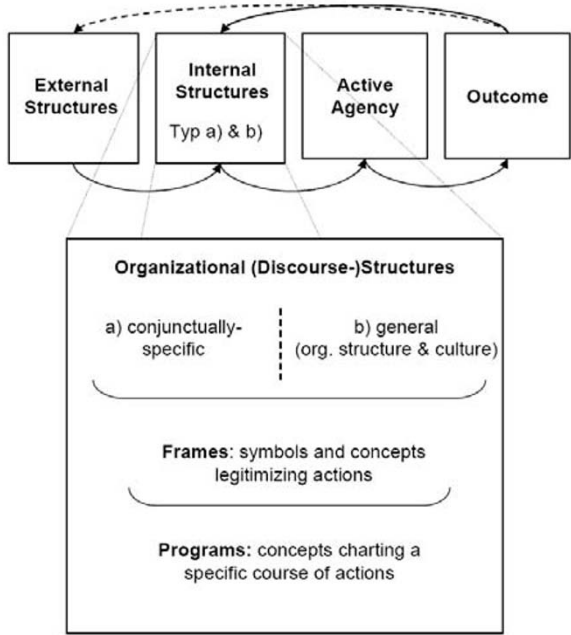


Fig. 2. Organizational discourse structures

4 Munich’s Migration Process and Discourse

The following description of the FOSS migration process in Munich is a radically condensed version of a much more detailed narration provided in [8].

4.1 The process: emergence of radical change

One core concept of path dependency theory in a narrow sense³ is that of “small events”: A series of small, partially unrelated or even stochastic events sets in motion a self-reinforcing process that eventually leads from a contingent state at the

³ “Narrow” is meant in the tradition of David [6] and Arthur [3] and compared to scholars using “path” and “path dependency” only as a metaphor for the truism that “history matters”. For an extensive discussion, see Dobusch and Schüßler [9].

beginning to a state of ex-ante unpredictable lock-in [7, 26,]. The case of Munich demonstrates that small events can also play a decisive role in the early stages of a path breaking process.

The politician – a private user of the proprietary “Ami Pro” – who set the ball rolling by suggesting the search for alternatives to Microsoft (MS) Office was only a backbencher (“Hinterbänkler”) in the city council, which he left soon after his motion. The first reaction of the IT officials was an attempt to turn the whole issue done by a simple product comparison between MS Office and several alternatives (see Table 4).

Table 4. Product comparison prepared by the central IT department for the responsible council committee's meeting in Nov. 2001

Advantages	Disadvantages
StarOffice is Open Source Software	Re-working in existing documents necessary because of
Lower purchase costs	- partial lack of import/export
End of dependence on Microsoft	- different macro languages
Cross platform applicability (StarOffice)	- different object models
	New product line needs requires more training and, therefore, leads to higher training costs
	There is no in-house training personnel
	Insufficient online-help (StarOffice)
	StarOffice/SmartSuite data formats are mostly not suitable for data exchange. Documents would have to be handed on in MS Office or RTF format.
	The already deployed – and paid for – MS Office products would have to be replaced all over although they are often not even amortised (to ensure readability in internal data exchange)
	No mail-client (SmartSuite)

Dissatisfied with their IT official's analysis, the political members of the committee demanded a second, more in-depth evaluation with special focus on economic efficiency, also taking into consideration the desktop operating system, as Microsoft had already cancelled its support for the Windows version in use. The following expert's advice [28] then convinced the IT staff of the viability of a FOSS solution. When this result was presented to the council committee, the situation surprisingly was the opposite way round compared to autumn 2001, as one of the officials describes:

“When, for the first time, we proposed Linux on the basis of the external study in autumn 2002 in the IT committee [...] we got a lot of stick ((laughs)) [...] and I thought we don't even need to continue, we slunk off with our tails between our legs.”

At the same time, the results of the “Client Study” and its recommendation to migrate alerted Microsoft – up to the CEO level: In April 2003, Microsoft CEO Steven

Ballmer interrupted his holidays in Switzerland to visit Munich's mayor. But, again, an attempt to turn down the migration plan even strengthened the momentum behind it: The amount of media coverage skyrocketed, thus increasing pressure on politicians to resist to the monopolist's power. As another consequence of the increased public interest, many firms and other municipalities encouraged Munich's officials and politicians to stay on their way and offered knowledge exchange. Additionally, Microsoft's move also motivated its competitors IBM and NovellSuse to upgrade their offers of assistance during a potential migration process. In May 2003, after weeks of heated seesaw changes in the calculation of the different alternatives, the council made the principle decision in favour of a migration to FOSS and authorised its IT department to work out a detailed draft for the project.

Consciousness of the actual costs, complexity and efforts connected with this decision only rose incrementally along the following migration process. What had started as a search for an alternative office software suite in 2001 had ended as a complete restructuring of the municipality's IT organization and processes:

- After the migration, only one standardized and centrally developed operating system version replaced the several different Windows versions.
- As a consequence of the standardized client, many prior decentralised tasks and routines (e.g. operating system set up⁴, administration and configuration) were going to be fulfilled by a central client team.
- In introducing a new standardized tool to manage office forms and master documents called "Wollmux"⁵, the city's corporate design guidelines were to be reinforced and – for the first time in history – to be implemented uniformly in the whole municipality.
- The change of the operating system gave reason for a general consolidation of the municipality's diverse software landscape, reducing the variety of applications in use.

Summing up this short description, there are at least three remarkable aspects of Munich's odyssey from Windows to FOSS (see also Figure 3): First, technology adoption and organisational change – especially concerning different degrees of organisational (de)centrality – are deeply intertwined and reciprocally enforcing. Second, roles and preferences of individual actors change during the process, converting some of them from opponents into proponents and vice-versa. It is, therefore, impossible – or at least, misleading – to simply reduce individual attitudes towards the adoption of Free and Open Source Software to their job functions as it is tried by Alexy and Henkel [1], and underlines the importance of applying a process perspective in researching adoption decisions. Third, external advice and

⁴ Although all departments had adopted the same operating system (Windows NT), its set up in terms of administration tools, settings, and support software varied.

⁵ "Wollmux" is a neologism combining the German "eierlegende Wollmilchsau" (colloquial for "all-in-one device suitable for every purpose") and the name of the Linux mascot "Tux".

interventions play an important catalytic role in the process, but at the same time are only “perturbations” [17] of overall internal organizational dynamics.

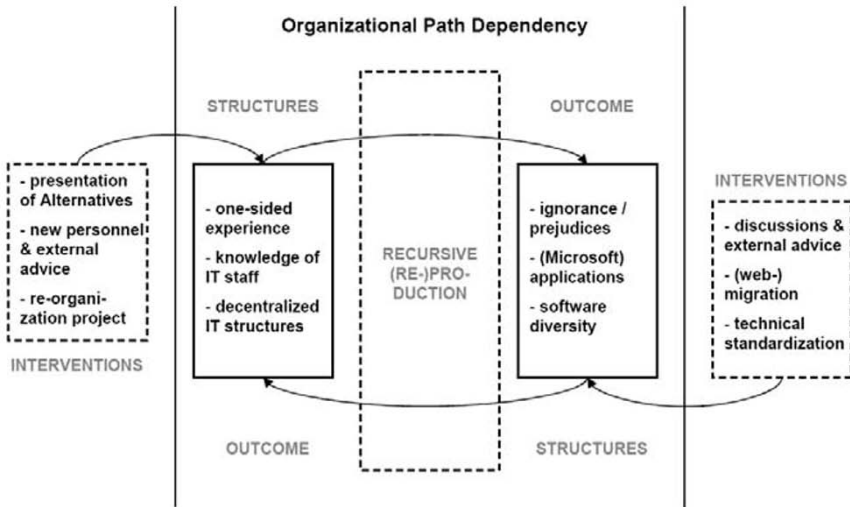


Fig. 3. Recursive re-production of organizational software paths and respective interventions

4.2 The discourse: coalitions of actors and issues

Table 5 depicts relative frequencies of dominant issues raised by actors in different organizational subsystems – political administration, central and decentral IT departments – on the basis of inductively generated coding categories [18]. It distinguishes thereby between legitimizing frames and programs charting a specific course of action [4, 5] both of which were – depending on the respective context – predominantly coded as either pro- or anti-migration⁶.

In Munich, two different organizational camps applied different frames to legitimize the same FOSS migration project and thus build what Hajer [14] calls a “discourse coalition”. On the one hand, politicians stressed frames referring to a diffuse “anti-monopoly” ideology and the particular responsibility as a public administration. The central IT officials, on the other hand, framed the introduction of FOSS above all as the (looked-for) opportunity to re-centralize the municipality’s organizational IT structure; for them – as the only subgroup – switching gains exceed switching efforts. Together, these two camps generate enough momentum [15] to

⁶ Except for some unidimensional coding categories, most categories aggregate two antithetic codes that cover statements referring either to barriers and downsides or drivers and advantages of a migration to FOSS. A category was classified as “predominantly pro- or anti-migration” if there was a difference larger than unity between the two.

overcome the strong innovation barriers of the desktop software markets. Together, these two camps generate enough momentum to overcome the strong barriers to FOSS introduction that materialized also in resistance in decentralized IT departments. Those opposed the migration as their expert’s knowledge – gained over years of “learning by doing” [2] – depreciated rapidly under a new operating system because of the „creative destruction“ of skills and knowledge in every technological innovation process [27].

Table 5. Dominant Frames and Programs in Munich’s Migration Discourse

	Frames (Top 3 out of 15 categories with a total of 439 codings)	Programs (Top 3 out of 12 categories with a total of 486 codings)
Political administration	18,68 % anti-monopoly(***) 17,58 % (overall) profitability 14,29 % public administration issues	27,14 % discussions 15,71 % external advice 12,86 % migration complexity(*)
Central IT department	25,98 % switching gains & efforts 18,63 % organisational structures(*) 15,20 % (overall) profitability	27,52 % migration strategy 24,03 % applications (incl. macros)(**) 9,30 % organisational know-how(*)
Decentral IT departments	26,39 % organisational structures(*) 22,92 % switching gains & efforts(*) 14,58 % expectations & uncertainty	17,09 % organisational know-how(*) 16,46 % migration strategy 15,19 % migration complexity(*)

(*) predominantly coded anti-migration
(**) unidimensional anti-migration code
(***) unidimensional pro-migration code

Looking at the categories that predominantly cover program aspects in the migration discourse (see right column in Table 5) one can see several facets more clearly: First, migration to FOSS is hard work and requires a lot of effort, in particular in cases of large-scale, rather complete migration projects like the one in Munich: Especially in both IT-departments categories that are predominantly coded anti-migration prevail and reflect the efforts necessary to migrate hundreds of special purpose applications and thousands of macros, forms and templates. Second, the program categories mirror the interventions depicted in Figure 3 and hence give an impression of the great variety of tasks necessary in such a migration project. Third, the again anti-migration position of the officials in the decentralized IT departments corresponds with the fact that they have to carry most of the migration burdens. Many of the migration tasks such as for example altering macros or forms means work, which at least in parts is additional to day-to-day routines.

Compared to other prominent and large-scale FOSS migration examples like the French car manufacturer PSA Peugeot⁷ or police forces in France and

⁷ <http://ec.europa.eu/idabc/en/document/6674/469>, accessed 27 February 2008.

Niedersachsen/Germany⁸, Munich's migration enterprise is illuminative for at least two reasons: First, migration complexity and efforts seem to grow disproportionately if not only significant parts as in the case of PSA Peugeot but the complete software environment is to be changed in a rather short period of time. Second, migration complexity and efforts seem to grow disproportionately with functional and hence software diversity in an organization. As opposed to police force cases in France and Germany where only few special purpose applications have to be migrated and the functional diversity is relatively low, in large municipal administrations the situation is the other way round. This in turn led at least in Munich to the need for organizational re-structuring which complicated the migration process and vice-versa.

5 Conclusions

Not only because of a municipality being a (at least: partially) political organization, the decision to (not) adopt FOSS in organizational contexts is always a political one: In organizations there will be winners and losers of migration processes as switching one's desktop software environment affects (complementary) organizational variables such as degree of decentralization or individual and organizational know-how. As a consequence, future research of FOSS adoption processes should include both looking at the (changes in the) migration discourse and at its organizational contextualisation. This study can only be a first step into this direction.

Acknowledgments

I wish to thank Jörg Sydow, Sigrid Quack, Jochen Koch, Arndt Sorge and Elke Schübler as well as all other members of the doctoral program "Research on Organizational Paths" at Freie Universitaet Berlin and the participants at the First Graz Schumpeter Summer School 2007 for their valuable comments.

References

- [1] Alexy, O. and J. Henkel (2007). Promoting the Penguin: Who is Advocating Open Source Software in Commercial Settings? Working Paper. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=988363. Accessed 16 October 2007.
- [2] Arrow, K.J. (1962). The Economic Implications of Learning By Doing. *The Review of Economic Studies*, 29(3), 155-173

⁸ <http://ec.europa.eu/idabc/en/document/7343/528>, accessed 27 February 2008 and <http://www.heise.de/newsticker/meldung/40374/> accessed 14 December 2008.

- [3] Arthur, B. (1989). Competing Technologies, Increasing Returns, and Lock-In by Historical Events. *The Economic Journal*, 99, 116-131
- [4] Campbell, J.L. (1998). Institutional analysis and the role of ideas in the political economy. *Theory and Society*, 27, 377-409
- [5] Campbell, J.L. (2004). *Institutional Change and Globalization*. Princeton: Princeton University Press.
- [6] David, P.A. (1985). Clio and the economics of QWERTY. *American Economic Review*, 75 (2), 332-337
- [7] David, P.A. (2000). Path Dependence, its critics and the quest for historical economics. Stanford Working Paper Series. <http://www-econ.stanford.edu/faculty/workp/swp00011.pdf>. Accessed 16 April 2007.
- [8] Dobusch, L. (2007). Adopter Innovation in Network Markets: The Case of Linux in Munich. Paper presented at the SASE Annual Meeting, Kopenhagen 28-30 June 2007, Network on Knowledge, Technology and Innovation.
- [9] Dobusch, L. and E.S. Schuessler (2007). From Storytelling to Theory: Unlocking Path Dependency from Metaphorical Usage. Paper presented at 23rd EGOS Colloquium, Vienna 5-7 July 2007, Sub-theme 14: Path dependencies and beyond.
- [10] Farrell, J. and G. Saloner (1986). Installed Base and Compatibility: Innovation, Product Preannouncements, and Predation. *The American Economic Review*, 76 (5): 940-955
- [11] Feller, J., B. Fitzgerald, S.A. Hissam, and E.R. Lakhani (Eds./2007). *Perspectives on Free and Open Source Software*. MIT Press. <http://mitpress.mit.edu/books/chapters/0262562278.pdf>. Accessed 28 March 2007.
- [12] Giddens, A. (1984). *The Constitution of Society: Outline of the Theory of Structuration*. Cambridge: Polity Press.
- [13] Gosh, R.A., B. Krieger, R. Glott and G. Robles (2002). Free/Libre and Open Source Software: Survey and Study. Part 2B: Open Source Software in the Public Sector: Policy within the European Union. International Institute of Infonomics, University of Maastricht. http://www.flossproject.org/report/FLOSSFinal_2b.pdf. Accessed 15 October 2007.
- [14] Hajer, M.A. (1993). Discourse Coalitions and the Institutionalization of Practice: The Case of Acid Rain in Great Britain. In Fischer, F. and J. Forester (Eds.): *The Argumentative Turn in Policy Analysis and Planning* (pp. 43-76). Durham and London: Duke University Press.
- [15] Hughes, T.P. (1983). *Networks of Power: Electrification in Western Society 1880-1930*. Baltimore: Johns Hopkins University Press.
- [16] Mayring, P. (2003). *Qualitative Inhaltsanalyse: Grundlagen und Techniken.* Weinheim and Basel: Beltz
- [17] Maturana, H.R. and F.J. Varela (1987). *The Tree of Knowledge: The Biological Roots of Human Understanding*. Boston: Shambhala.
- [18] Miles, M.B./Huberman, A.M. (1994). *Qualitative Data Analysis: An expanded Sourcebook*. Second Edition. Thousand Oaks: Sage
- [19] Nagler, M. (2005). Migration of the German City of Schwäbisch Hall. <http://ec.europa.eu/idabc/servlets/Doc?id=19734>. Accessed 12 December 2007.
- [20] Orlikowski, W.J. (1992). The Duality of Technology: Rethinking the Concept of Technology in Organizations. *Organization Science*, 3 (3), 398-427
- [21] Orlikowski, W.J. (2000). Using Technology and Constituting Structures: A Practice Lens for Studying Technology in Organizations. *Organization Science*, 11 (4), 404-428
- [22] Rossi, B., B. Russo and G. Succi, G. (2007). Open Source Software and Open Data Standards as a form of Technology Adoption: a Case Study. In Feller, J., B. Fitzgerald, W.

- Scacchi and A. Sillitti (Eds.): IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, 325-330
- [23] Shapiro, C. and H.R. Varian (1999). *Information Rules: a strategic guide to the network economy*. Cambridge (MA): Harvard Business School Press
- [24] Stones, R. (2005). *Structuration Theory*. Houndmills et al.: Palgrave Macmillan:
- [25] Strauss, A.L. and J. Corbin (1990). *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Newbury Park (CA): Sage
- [26] Sydow, J., G. Schreyögg, G. and J. Koch (2005). *Organizational Paths: Path Dependency and Beyond*. Paper presented at the 21st EGOS Colloquium, June 30 - July 2, 2005, Berlin, Germany
- [27] Tushman, M.L. and P. Anderson (1986). *Technological Discontinuities and Organizational Environments*. *Administrative Science Quarterly*, 31, 439-465
- [28] Unilog Integrata (2003). *Client Studie der Landeshauptstadt München: Kurzfassung des Abschlussbereichs inklusive Nachtrag*. Stand 02.07.2003. <http://www.muenchen.de/limux>. Accessed 26 February 2008.
- [29] Varian, H.R., J. Farrell, and C. Shapiro (2004). *The Economics of Information Technology: An Introduction*. Cambridge (UK): Cambridge University Press.
- [30] Varian, Hal R. and Carl Shapiro (2003), *Linux Adoption in the Public Sector: An Economic Analysis*, <http://www.sims.berkeley.edu/~hal/Papers/2004/linux-adoption-in-the-public-sector.pdf>. Accessed 28 October 2005.
- [31] von Hippel, E. and G. von Krogh (2003). *Open Source Software and the „Private-Collective“ Innovation Model: Issues for Organization Science*. *Organization Science*, 14(2), 209-223
- [32] Wichmann, T. (2002). *FLOSS Final Report – Part 1: Use of Open Source Software in Firms and Public Institutions – Evidence from Germany, Sweden and UK*. Berlin: Berlecon Research.
- [33] Williamson, O.F. (1985). *The Economic Institutions of Capitalism*. New York: The Free Press.
- [34] Yin, R.K. (1994). *Case Study Research. Design and Methods*. Thousand Oaks: Sage.

The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation

Ioannis Samoladas¹, Georgios Gousios²,
Diomidis Spinellis², and Ioannis Stamelos¹

¹ Department of Informatics, Aristotle University of Thessaloniki,
541 24, Thessaloniki, Greece

{ioansam, stamelos}@csd.auth.gr

² Department of Management Science and Technology,
Athens University of Economics and Business, 104 34, Athens, Greece
{gousiosg, dds}@aueb.gr

Abstract. Software quality evaluation has always been an important part of software business. The quality evaluation process is usually based on hierarchical quality models that measure various aspects of software quality and deduce a characterization of the product quality being evaluated. The particular nature of open source software has rendered existing models inappropriate for detailed quality evaluations. In this paper, we present a hierarchical quality model that evaluates source code and community processes, based on automatic calculation of metric values and their correlation to a set of predefined quality profiles.¹

Keywords: Quality models, automated measurement, software metrics

1 Introduction

One of the main concerns of software engineering is the production of high quality software systems and thus software quality evaluation has always been a critical task for software professionals. IT managers often face the problem of evaluating software in order to decide whether it is suitable for their needs. Additionally, software houses perform evaluations on the software they develop to decide whether it has matured enough to be deployed. Evaluations are based on software models that define and measure software quality, usually by combining software metrics and experts' opinions. The advent of free, libre and open source software (OSS) has rendered the traditional quality evaluation models non applicable to some extent, as

¹ This work was partially supported by the European Community's Sixth Framework Programme under the contract IST-2005-033331 "Software Quality Observatory for Open Source Software (SQO-OSS)".

Please use the following format when citing this chapter:

Samoladas, I., Gousios, G., Spinellis, D. and Stamelos, I., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 237–248.

they cannot be tuned to reflect OSS development practices and therefore cannot be used to evaluate both the software and the community as a whole.

In this paper, we present a novel software quality evaluation model, specifically targeted to OSS. The SQO-OSS model was constructed to support an automated software evaluation system; its variables are mainly metric-oriented while human intervention is minimal. Additionally, our model evaluates all aspects of OSS development, both the product (code) and the community. The evaluation weights and criteria can be tuned by the evaluator, while a set of predefined profiles that cover basic evaluation cases are offered.

The remainder of the paper is organized as follows: In Section 2, we present related work in the area of both traditional and open source software quality evaluation; in Section 3, we present the SQO-OSS quality model definition and the evaluation process. Section 4 presents an application of a part of the model on three example open source projects. The paper concludes with a description of the Alitheia system, the host system for software metric calculations, as well as our plans for future work.

2 Related Work

Since researchers started investigating the issue of quality in software systems, they employed specific models to express it. Models usually decompose quality into a hierarchy of criteria and attributes². These hierarchical models lead to metrics at their lowest level. Metrics are directly measurable attributes of software and they are used to express certain aspects of the product that affect quality [1]. Examples of traditional software quality models are the McCall and Boehm's models [1], the more widely accepted ISO/IEC 9126 model [2], and its more recent implementation by SQuARE, the ISO 25000:2005 [3].

The adoption of OSS in many organizations has raised the issue of OSS quality evaluation. Due to the nature of OSS development where standard practices include open access to the source code, shared artifact repositories, peer review of committed code, asynchronous global development and lack of formal support, traditional software quality models may not be sufficient. An array of quality models specifically targeted to OSS development can be found in the literature, but most of them are either purpose specific or require significant human intervention.

The OSMM [4] model assumes that the quality of an open source project proportional to its maturity. The latter is decomposed into six constituents (*Product software, Support, Documentation, Training, Product integrations* and *Professional services*), each one having some weight. The evaluator assigns a score to each element and the final evaluation mark is the weighted sum of the scores. Although

² Throughout the paper the terms criterion and attribute (sub-criterion and sub-attribute) are used interchangeably.

OSMM is simple and thus easy to apply, it is often criticized for not taking into account some important software artifacts, such as the source code itself.

The Open Business Readiness Rating (OpenBRR) [5] defines a model and a process for evaluating OSS, with particular emphasis on attributes interesting to businesses. OpenBRR uses a variety of high-level criteria for evaluation, such as functionality, operational software characteristics, support and service and adoption and development process. The assessment process involves defining a reference application and through it identifying a set of characteristics (and weights for them) that are desirable in the evaluated applications. The evaluation result is extracted by assigning grades to each characteristic and averaging the results from the evaluators. While the OpenBRR method is a step forward from OSMM through the inclusion of the community process in the evaluation, the notion of the reference application has been criticized as a major drawback. Furthermore, the evaluation itself is highly subjective, while the overall process seems complicated, offering very little prospect for automation.

The Qualification and Selection of Open Source Software (QSOS) [6] is another open source evaluation model. The evaluation process is done in four iterative phases. Phase one is the definition of the evaluation factors. The second phase involves the collection of information from the open source community and the construction of an identity card for the evaluated software. The quality criteria are then scored in a range from zero to two according to specific guidelines provided by the methodology. Phase three is the definition of the selection criteria according to user's needs and constraints. The last phase is the identification of the software that fulfills user requirements and more generally compares software from the same family. Like OpenBRR, QSOS offers a tool that supports the evaluation process. Although QSOS scoring guidelines allow for objective results among users, the whole process is not flexible enough and difficult to handle.

The SQO-OSS quality model distinguishes from existing open source quality models in various ways:

1. The SQO-OSS model was constructed with a focus to automation, while the rest of the models require heavy user interference and lack automation of metrics collection.
2. The SQO-OSS model is the core of a continuous quality monitoring system and automatic metrics collection guarantee that assessments are made with relatively recent data.
3. The SQO-OSS model does not evaluate functionality. Functionality assessment requires the evaluator to play an important role in the assessment process and thus introduces subjectivity. The SQO-OSS model focuses on fundamental aspects of OSS quality, namely OSS project maintainability, reliability and security.

4. The SQO-OSS model focuses on source code. Source code is the single most important product of a software development project and its quality must play a significant role in determining the final assessment of the product.
5. The SQO-OSS model also considers the open source community. However, it takes into account only those community factors that can be measured automatically.
6. As the evaluation must necessarily take into account the evaluator's point of view, we allow the user to intervene in the measurement-based evaluation process by modifying category profiles.

3 The SQO-OSS Quality Model

We can generally assume that an evaluation process can be divided into two phases, the definition of the evaluation model and the definition of the measurement process. In our case, each phase includes two distinct steps:

Phase One: *Definition of the evaluation model*

1. Definition of the model criteria (attributes and sub-attributes).
2. Definition of metrics.

Phase Two: *Definition of the aggregation method*

1. Definition of the evaluation categories
2. Definition of the profiles of those categories

Phase one represents the work done in order to define the evaluation framework, applying our notion of quality, the definition of the quality criteria and the metrics that measure these criteria. The second phase represents the data collection part and the aggregation of the measurement results in order to reach an outcome for the quality of the artifact under evaluation. At this point we have to lay more emphasis on the fact that throughout this process, automation was the main concern. We wanted a model that can be applied automatically and on a fair amount of projects, fed continuously with data from the SQO-OSS observatory. Thus we tried to focus on quality attributes that can be measured with minimum human interference. For example usability involves extensive human interaction so it was not chosen as a quality attribute for our model. The same approach was followed for metrics selection.

3.1 Model definition

Prior to starting the construction of our model, we took into account that the quality and the health of an OSS project depends on the quality of its source code base and that of the community built around it. In order to measure these two aspects of quality and construct the SQO-OSS quality model we used a simplified version of the Goal-Question-Metric (GQM) process [7].

For the first part, we formulated our first goal, “*analyze the source code of an open source project*”. This analysis has to be done in order to characterize code quality with respect to its maintainability, reliability and security. So, for this goal we formulated the question “*How is source code quality measured?*”. We then formulated the question as “*How is maintainability, reliability and security measures?*” In turn, each one of these aspects of source code quality is a small goal itself with its own questions. We kept on formulating questions iteratively until we reached a level where an attribute can be measured using straightforward metrics, i.e. without any usage of compound metrics.

For example, in order to measure *Maintainability*, we chose to follow its definition in the ISO/IEC 9126 quality model: *Analyzability, Changeability, Stability, Testability*. To measure each one of these sub-attributes we used direct source code metrics. In order to differentiate between structured and object oriented programming we used different definitions of *Maintainability* attribute for these two programming paradigms. Metrics selection was a difficult part, since software engineering metrics research is a very active topic and many researchers express their concerns on various metrics. For our own model we chose to select only widely acceptable metrics and metrics that have been validated extensively [8]. For an extensive review on metrics please refer to [9] and [1].

In a similar fashion, we constructed a hierarchical view of our quality model as a tree. The root represents the overall quality model, the next two nodes the source code and community quality, while the leaves represent the metrics. After constructing the initial model, we uploaded the model on the wiki page of our project asking from our consortium partners to review it and comment on it. Our partners come from the OSS community (developers, users), academia and companies specializing on OSS development and support. Additionally, we also asked them to change the model (both model leaves and respective metrics) and justify their opinion. The only thing stressed to the partners was the importance of automatic metric collection. The history facility of the wiki allowed us to review the changes, discuss them with the partners and finalize the model. A tree view of the model is presented in Figure 1, while the metrics selected for the evaluation of the selected criteria (the leaves on the tree view) can be seen in Table 1. At this point, we should mention that our system allows partial evaluation of a product, i.e. evaluation of a single attribute like *Testability*. Thus, we have used the same metrics in more than one attribute.

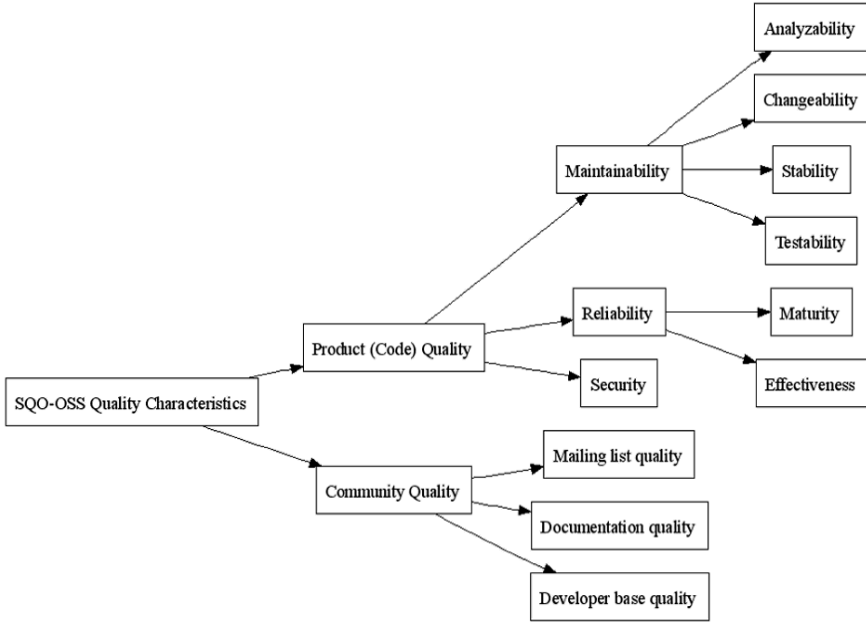


Fig. 1. The SQO-OSS quality model.

Table 1. Metrics for criteria of Product (Code) Quality and Community Quality

Attribute	Metric
Analyzability	Cyclomatic Number
	Number of statements
	Comments frequency
	Average size of statements
	Weighted methods per class (WMC)
Changeability	Number of base classes
	Class comments frequency
	Average size of statements
	Vocabulary frequency
	Number of unconditional jumps
	Number of nested levels
Stability	Coupling between objects (CBO)
	Lack of cohesion (LCOM)
	Depth of inheritance tree (DIT)
	Number of unconditional jumps
	Number of entry nodes
	Number of exit nodes

	Directly called components
	Number of children (NOC)
	Coupling between objects (CBO)
	Depth of inheritance tree (DIT)
Testability	Number of exits of conditional structs
	Cyclomatic number
	Number of nested levels
	Number of unconditional jumps
	Response for a class (RFC)
	Average cyclomatic complexity per method
	Number of children (NOC)
Maturity	Number of open critical bugs in the last 6 months
	Number of open bugs in the last six months
Effectiveness	Number of critical bugs fixed in the last 6 Months
	Number of bugs fixed in the last 6 months
Security	Null dereferences
	Undefined values
Mailing list	Number of unique subscribers
	Number of messages in user/support list per month
	Number of messages in developers list per month
	Average thread depth
Documentation	Available documentation documents
	Update frequency
Developer base	Rate of developer intake
	Rate of developer turnover
	Growth in active developers

3.2 Evaluation Process

In order to evaluate the quality of an open source project, we have to combine all these measurements in one single view to obtain an evaluation result, i.e. we have to aggregate the measurements. For this, we used the profile-based evaluation method described in detail in reference [10]. Most evaluation methods presented in the literature use a weighted average sum function as their aggregation method. A drawback for applying this aggregation method on our model is that it uses measures with interval scales, while our goal was to provide results in an ordinal scale (such as *good*, *fair* or *poor*). The method we selected allows us to combine all kind of measurements, based on either ordinal or interval scales.

In order to apply an ordinal scaling aggregation method, we must first decide how many categories of evaluation ranking are required. In reference [10], Morisio et al. discuss that the ideal number of evaluation categories is between three and five. Based on that, for our model, we used four categories: **Excellent** (E), **Good** (G), **Fair** (F) and **Poor** (P) (or as an ordinal scale $E > G > F > P$). Having four categories, the aggregation method requires the definition of three profiles, each one for the first three categories (E , G and F). If an artifact cannot be fitted into one of these three categories, then it is automatically categorized as poor. The profiles represent the least measurement values required for each category and they are defined separately for each composed criterion of the model. Then each criterion is further decomposed into its sub-criteria (hierarchically, according to the quality model) and each decomposed criterion has its own profile. For the quality model leaves, which consist of an array metrics used to assess their parent criterion, we use a vector of numbers that is constructed by the application of each specific metric to the assessed artifact. We used the thresholds indicated by the literature [11, 12, 13] to correlate the measurement value vectors to the profiles we had developed (see Table 2).

Decomposing root attributes of our quality model into more fine-grained criteria and then down to metrics entails that profiles correspond to each decomposed criterion. Thus, in order to characterize the product quality of a product as “Excellent”, Maintainability, Reliability and Security must be also characterized as “Excellent”. Table 2 shows that a software component that scores “*Excellent*” in Maintainability must score “*Excellent*” in the Analyzability sub-attribute, too. This, in turn, means that the corresponding metrics applied to the evaluated component must return values equivalent to or higher than the vector [4,10,0.5,2]. Similar profiles correlating metric values to profiles can be applied to the rest of the criteria. Due to space limitations, Table 2 shows only the metrics values for Analyzability and Changeability; however, similar thresholds exist for the rest of criteria. Users of the model can modify the profiles according to their needs, e.g. a security aware user may define higher default values for each of the profiles. Additionally, the method allows usage of weights on the various metrics, but such practice is not recommended, as we assume that all metrics are of equal importance.

The aggregation process is done with the use of specific outranking relations iteratively with all the given profiles. The outranking relations express our decision of comparing the artifact with the profiles. Thus, an artifact x is considered to be *at least as good as* the y profile if and only if the “weighted” majority of the criteria agree so. If a set of tests agree, which represent the strength to be reached in order for an artifact to be categorized in a category A then x is assigned to A . The method also allows for two kinds of assignments in categories. The first is the pessimistic assignment representing the *at least as good as* relation (project x is *at least as good as* profile y). The second is the optimistic assignment, which identifies the profile which is surely worse than x and assigns x to the previous one (for example if x is strictly worse than E then it is assigned to G). If the two assignments coincide, then we are sure about our decision, otherwise it is the evaluator’s decision which of the two assignments will be adopted. The mathematical foundations and the actual

procedure of the aggregation process is presented in reference [10]. The method used here is different than Analytic Hierarchy Process, a widely used aggregation method, which requires ratio scales on all measures, a requirement that is not fulfilled in our case. Moreover, as already mentioned earlier, having a weighted average sum function as an aggregation method forces us to use measures with interval scales; our goal was to provide results on an ordinal scale, a feature that is provided with the method presented.

An example of the aggregation process is presented in the next section.

Table 2. Profiles for criteria Analyzability, Changeability, Stability and Testability

Composed Criterion	Criterion	Profile E	Profile G	Profile F	Scale
Analyzability	Cyclomatic number	4	6	8	Less is better
	Number of statements	10	25	50	Less is better
	Comments frequency	0.5	0.3	0.1	More is better
	Average size of statements	2	3	4	Less is better
Changeability	Average size of statements	2	3	4	Less is better
	Vocabulary frequency	4	7	10	Less is better
	Number of unconditional jumps	0	0	1	Less is better
Stability	Number of nested levels	1	3	5	Less is better
	Number of unconditional jumps	0	0	1	Less is better
	Number of entry nodes	1	2	3	Less is better
	Number of exit nodes	1	1	1	One is better
Testability	Directly called components	2	5	7	Less is better
	Number of exits of conditional structs	0	1	4	Less is better
	Cyclomatic Number	4	6	8	Less is better
	Number of nested levels	1	3	5	Less is better
	Number of unconditional jumps	0	0	1	Less is better

4 Evaluation Example

In this section, we present an example of application of the SQO-OSS quality model. For our example, we evaluated the source code of the CVS versioning system, the interpreter of the Perl programming language and the C files of the FreeBSD operating system. Table 5 shows the performance of these projects according to the various measurements. All measurements were performed using the metrics extracted from the application of the CScout refactoring browser [14] on the evaluated projects. According to the measurements, two out of our three projects

scored well in the maintainability criterion. A direct interpretation of these results according to the proposed model is that CVS and FreeBSD are *at least as good as* the metrics thresholds for the **Good** profile, while Perl is *at least as good as* the thresholds for **Fair**.

A careful examination of the results reveals details of the performance of these projects in various sub-attributes of maintainability. All three projects achieve high marks in the changeability metrics and a good level of analyzability, perhaps due to their development model. Stability and testability are fair, but close to the metric thresholds we set for each respective criterion. Taking into consideration that the thresholds used in our example are relatively strict, the results are encouraging even for these two factors. Apart from providing predefined thresholds, our method has the benefit of presenting the results in a way that enables the developer to focus on measurements that are really interesting to him and also to receive both coarse and fine grained information.

Table 5. Example measurements of projects CVS, Perl and FreeBSD

Composed Criterion	Project evaluation		
	CVS	Perl	FreeBSD
Analyzability	[5.63, 37.8, 0.34, 1.89]	[3.0, 36.17, 0.08, 1.02]	[3.82, 29.99, 0.12, 1.99]
<i>Evaluation</i>	<i>Good</i>	<i>Fair</i>	<i>Excellent</i>
Changeability	[1.89, 2.91, 0.24, 1.13]	[1.02, 2.42, 0.28, 0.53]	[1.99, 2.87, 0.51, 0.86]
<i>Evaluation</i>	<i>Good</i>	<i>Excellent</i>	<i>Excellent</i>
Stability	[0.24, 3.28, 1.08, 4.49]	[0.08, 3.21, 0.78, 4.75]	[0.25, 3.99, 1.23, 4.10]
<i>Evaluation</i>	<i>Poor</i>	<i>Fair</i>	<i>Poor</i>
Testability	[0.57, 5.62, 1.13, 0.24]	[0.46, 3.0, 0.53, 0.08]	[0.55, 3.82, 0.86, 0.25]
<i>Evaluation</i>	<i>Good</i>	<i>Good</i>	<i>Good</i>
Maintainability	Good	Fair	Good

5 The Alitheia System

The quality model presented above serves as an automated decision support tool, integrated into the SQO-OSS system [15]. The SQO-OSS project aims to build a software quality observatory for OSS. For that purpose, we have developed Alitheia, a quality evaluation tool, and a web site with the results of the tool application on various OSS projects. The user is able to browse the product and process quality characteristics of the evaluated projects. The SQO-OSS quality model assists the user by incorporating the individual measurements in a comprehensive set of predefined quality profiles.

The Alitheia platform is an OSGi-based tool, targeted to the evaluation of software quality. It consists of a set of core services, such as accessors to project artifacts, job managers and relational data storage, and it is extensible through the use of plug-ins. Plug-ins can either implement basic software metrics or combine the

results from other plug-ins arbitrarily. In fact, the quality model is a compound plug-in in Alitheia. The system allows full automation of the quality evaluation process after the initial project registration. The core communicates to the world through a web services interface. Clients being developed include the aforementioned web site and an Eclipse plug-in.

6 Conclusion and Future Work

In this paper we presented a new open source software quality evaluation model. The model was constructed for use in the Alitheia system, as a measurement-based decision support system, therefore automation was one of the first priorities while constructing the model. Previous models developed for OSS evaluation require a substantial effort from the user regarding the rating of the software under evaluation, while the model presented here asks for limited user interaction. Apart from the model itself, the evaluation process facilitates a profile based evaluation algorithm that is different from the traditional weighted aggregation that most of the models use. The profiles used for evaluation can be altered by the evaluator if he decides it is needed so.

Our immediate plans are to empirically validate our model. In order to test our model we are collecting measurements from an array of OSS projects. To meet our goal, we are going to perform a user based validation. Our project consortium includes members of a large OSS project, namely KDE Desktop Environment. Partners from the KDE project will evaluate software against our model and their opinions will be tested against the results of our evaluation process. In addition we want to evaluate its predictability and accuracy regarding its ability to classify software according to its quality. These tests will also allow us to calibrate the threshold values of our profiles. Moreover, we will work towards testing the relationships between metrics and categories and try to identify trends between aspects of quality and metrics.

References

1. Norman Fenton and Shari Lawrence Pfleeger. *Software Metrics - A Rigorous Approach*. International Thomson Publishing, London, 1997.
2. ISO. Software engineering - Product quality - Part 1: Quality model, ISO/IEC 9126-1:2001. ISO Geneva, 2001.
3. ISO. Software engineering Software product Quality Requirements and Evaluation (SQuaRE) Guide to SQuaRE. ISO Geneva, 2005.
4. Bernard Golden. *Making Open Source Ready for the Enterprise, The Open Source Maturity Model*. Extracted From Succeeding with Open Source, Addison-Wesley Publishing Company, 2005.
5. Business Readiness Rating. Business readiness rating for open source. <http://www.openbrr.org>.
6. QSOS. Method for qualification and selection of open source software (qsos) version 1.6. <http://www.qsos.org>.

7. R. Van Solingen. The goal/question/metric approach. *Encyclopedia of Software Engineering*, 2:578-583, 2002.
8. Tibor Gyimothy, Rudolf Ferenc, and Istvan Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897-910, 2005.
9. Stephen H. Kan. *Metrics and Models in Software Quality Engineering*, 2nd Edition. Addison-Wesley Publishing Company, 2003.
10. Maurizio Morisio, Ioannis Stamelos, and Alexis Tsoukias. Software product and process assessment through profile-based evaluation. *International Journal of Software Engineering and Knowledge Engineering*, 13(5):495-512, 2003.
11. NASA and SATC. Recommended thresholds for non-oo languages. http://satc.gsfc.nasa.gov/metrics/codometrics/non_oo/thresholds/index.html.
12. NASA and SATC. Recommended thresholds for oo languages. <http://satc.gsfc.nasa.gov/metrics/codometrics/oo/thresholds/index.html>.
13. Saida Benlarbi, Khaled El Emam, Nishith Goel, and Shesh Rai. Thresholds for object-oriented measures. In *Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, 2000.
14. Diomidis Spinellis. Global analysis and transformations in preprocessed languages. *IEEE Transactions on Software Engineering*, 29(11):1019-1030, November 2003.
15. Georgios Gousios, Vassilios Karakoidas, Konstantinos Stroggylos, Panagiotis Louridas, Vasileios Vlachos, and Diomidis Spinellis. Software quality assesment of open source software. In *Proceedings of the 11th Panhellenic Conference on Informatics*, May 2007.

An Open Integrated Environment for Transparent Fuzzy Agents Design

Giovanni Acampora and Vincenzo Loia

Department of Mathematics and Computer Science, University of Salerno
via Ponte don Melillo, 84084 Fisciano, Salerno, Italy
{gacampora,loia}@unisa.it

Abstract. Recently, computational agents received significant attention in computer science research community. In fact, intelligent agents is a powerful artificial intelligence technology showing considerable promise as a new paradigm for mainstream software development and able to offer new ways of abstraction, decomposition, and organization that fit well with our natural view of the world. However, despite their promise, intelligent agents are still scarce in the market place. A key reason for this is that developing intelligent agent software requires significant training and skill. Artificial Intelligence methodologies and computer networking tools represent the necessary basic knowledge to design and implement advanced agents oriented systems. This papers introduces an integrated development environment supporting the agents developers to design fuzzy-based agents in a simple and fast way. Proposed framework has been realized by integration of theoretical methodologies as fuzzy logic and labeled tree, together with OSS tools as JaxMe2.

1 Introduction

Multi-Agent systems (MASs) consist of a large number of computational agents, interconnected by communication devices, that seamlessly work together in order to achieve prefixed common goals. Different methodologies and techniques have to be exploited in order to design advanced multi-agent systems, ranging from social science, business models, network architectures, up to human interaction design. This methodologies integration allows to design and implement agents-based environment characterized by many features [1]. However, in spite of hard exploitation of computational agents in several domains of computer applications, MAS developers need of significant training, skills and experiences to design and develop advanced MAS-based frameworks. In fact, as previously depicted, MAS can be considered as a composition of different computer science backgrounds, mainly, Artificial Intelligence and Computer Networking. This paper introduces a novel design framework (FuzzyIDE), based on transparent fuzzy agents[2], which allows the system designers to express their ideas in fast and simple way achieving the properties of MAS without additional effort. In particular, this paper introduces a novel tree-based representation for fuzzy controllers allowing the designer to draw their ideas in a natural way and, at the same time, to split the whole controller in

Please use the following format when citing this chapter:

Acampora, G. and Loia, V., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succì; (Boston: Springer), pp. 249–255.

several subcontrollers by exploiting the subtree concept. Proposed framework has been realized by integration of theoretical and applicative methodologies as labeled tree and fuzzy control theory.

2 Transparent Fuzzy Control

Fuzzy control theory [3] can be considered as the most widely used application of fuzzy logic [4] [5] From a high level point of view, a Fuzzy Logic Controller (FLC) is an adequate methodology for designing and developing controllers capable of supplying high quality performance in environments characterized by high level of uncertainty and imprecision. However, in spite of these unquestionable advantages, the real design of FLCs is strongly depends upon hardware architecture which will implements the running version of designed controller.

2.1 FLC Labeled Tree

In this section we discuss the labeled tree structure[6] which will be exploited to define the FLC Labeled Tree, the tree-oriented data structure modeling FLCs and representing the core element of *Transparent Fuzzy Control*. A labeled tree is a tree where each node is associated with a label. More formally: Let Σ_V be finite alphabets of vertex labels and edge labels, respectively. Let V be a finite nonempty set of vertices, l a total function $l:V \rightarrow \Sigma_V$, E a set of unordered distinct vertices called edges, and a a total function $a:E \rightarrow \Sigma_E$. Then $G = (V, l, E, a)$ is a *labeled graph*. A *labeled tree* is a connected, acyclic labeled graph. Labeled tree labels may be constants, node variables (corresponding to any node value), or path variables (corresponding to any path). Constants corresponds to element, attribute and text values populating XML models. In details, nodes labeled with text values are called text nodes and they are always leaf nodes. Attribute nodes can have only one child node, a text node. Also, any two attributes of a given element cannot have the same label. Element nodes can have zero or more child nodes that can be elements, attributes, or text nodes. Usually, the labels related to element nodes, attribute nodes and text nodes are denoted, respectively, with n_i , a_i and $pcdata$. Then, in the case of XML document modeling: $\Sigma_V = \cup_i \{n_i\} \cup \cup_i \{a_i\} \cup pcdata$ and $\Sigma_E = \epsilon$. In order to represent an FLC through labeled tree structures, it is necessary to individuate the different components of a FLC, to map them on appropriate values for the labels n_i , a_i and $pcdata$ and, at the same time, to establish the opportune father-child relationships among different tree nodes. This analysis will result in an acyclic labeled graph, i.e, in a labeled tree. From a bottom-up point of view, a FLC can be view as a collection of fuzzy concepts and fuzzy rules composing, respectively, the fuzzy knowledge base and fuzzy rule base. Classic samples of fuzzy concepts are: temperature, pressure, speed, etc.. Each concept is defined, mainly, by means of: a *name*, a *universe of discourse* and a collection of fuzzy terms. Each fuzzy term can

be represented through a pair (*Fuzzy Set, Linguistic Expression*). By considering the a typical temperature fuzzy concept the following labels can be found:

$n_{FC_1} = FuzzyVariable, n_{FC_2} = n_{FC_3} = n_{FC_4} = FuzzyType \quad n_{FC_5} = TriangularShape$
 $a_{FC_1} = a_{FC_2} = name, a_{FC_3} = domainLeft$
 $a_{FC_4} = domainRight, a_{FC_5} = scale, a_{FC_6} = not, a_{FC_7} = param1, a_{FC_8} = param2, a_{FC_9} = param3$
 ...

with:

$$\Sigma_{VFC} = \{n_{FC_1}, n_{FC_2}, n_{FC_3}, n_{FC_4}, n_{FC_5}, a_{FC_1}, a_{FC_2}, a_{FC_3}, a_{FC_4}, a_{FC_5}, a_{FC_6}, a_{FC_7}, a_{FC_8}, a_{FC_9}, \dots\}$$

and

$$E_{FC} = \{(n_{FC_1}, a_{FC_1}), (n_{FC_2}, a_{FC_2}), (n_{FC_3}, a_{FC_3}), (n_{FC_4}, a_{FC_4}), (n_{FC_5}, n_{FC_5}), (n_{FC_6}, n_{FC_6}), (n_{FC_7}, a_{FC_7}), (n_{FC_8}, a_{FC_8}), (n_{FC_9}, a_{FC_9}), (n_{FC_{10}}, a_{FC_{10}}), (n_{FC_{11}}, a_{FC_{11}})\}$$

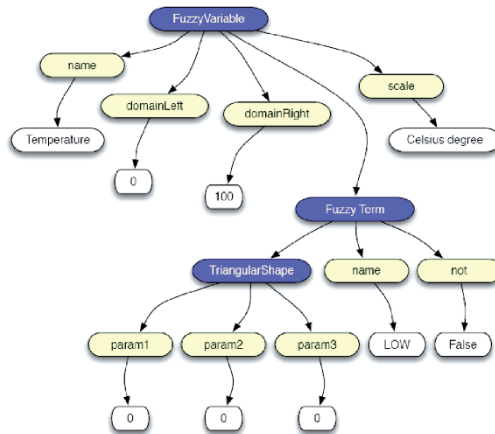


Fig. 1. Fuzzy Variable labeled tree

The *pcdata* information identifies the leaves of XML tree (white tree nodes), i.e., they are strongly depending upon real controller and, for this reason, they are not considered in previous analysis. Figure 1 shows the complete labeled tree derived from previous definitions. Finally, a fuzzy knowledge base can be modeled through a labeled tree whose labels are: $\Sigma_{KB} = \cup_i \Sigma_{VFC_i} \cup \dots$ with the following father-child relationships: $E_{KB} = \cup_i E_{FC_i} \cup \{(n_{KB}, n_{FC_1}), (n_{KB}, n_{FC_2}), \dots\}$, where \mathbf{I} is the label collection modeling i^{th} fuzzy concept and n_{KB} is a dummy root label joining together the different fuzzy concepts sub trees. At same way, focusing on fuzzy rule base, a fuzzy rule is a composition of fuzzy antecedent and consequent parts and,

recursively, the antecedents and consequents can be characterized by a sequence of fuzzy clauses, where, each fuzzy clause, is represented through a pair (*FuzzyVariable, FuzzyTerm*) (in Mamdani case). In short, the labeled tree modeling the fuzzy rule base is: $\Sigma_{RB} = \cup_i E_{FR_i} \cup \{(n_{RB}, n_{FR_1}), (n_{RB}, n_{FR_2}), \dots\}$, with the following father-child relationships: $E_{RE} = \cup_i E_{RB_i} \cup \{(n_{RB}, n_{RB_1}), (n_{RB}, n_{RB_2}), \dots\}$, where E is the label collection modeling i^{th} fuzzy rule and n_{RB} is a dummy root label joining together the different fuzzy rule sub trees. At the end each fuzzy controller can be modeled by means of a labeled tree populating by following labels: $\Sigma_{FLC} = \Sigma_{RB} \cup \Sigma_{KB} \cup \{n_i\}$ and $E_{FLC} = E_{KB} \cup E_{RB} \cup \{(n_{FLC}, n_{RE}), (n_{FLC}, n_R), \dots\}$, where n_{FLC} is a dummy node joining the fuzzy rule base and knowledge base.

3 A Novel Abstract Representation for Transparent Fuzzy Agent Design

The XML representation of FLC allows to model the controller in a human-readable and hardware independent way, i.e., through XML is possible to implement the same FLC on different hardware without additional design and implementation steps. The XML-based language modeling FLCs is named Fuzzy Markup Language(FML) and its fundamentals items, the tags and attributes, are the labels belonging to Σ set.

3.1 Fuzzy Markup Language Definition

The FLC labeled tree is an theoretical representation of a fuzzy logic controller. In order to move FLC tree model to a computer application context, it is necessary to define an opportune computer language grammar mapping the set of labels and relationships defined into the FLC labeled tree. *Document Type Definition* and *XML Schema* are used to define the FML grammar[7][8].

3.2 FLC Labeled Tree and FML in Multi-Agent Systems

The distribution of tasks and resources is both one of the major domains of multi-agent systems. In the FLC design could be very interesting and useful to distribute the fuzzy inference on different computing resources in order to speed up fuzzy computation and to localize rule subset 'near' their data sources. In order to allow this fuzzy distribution, FLC has to be considered as a set of distributed disjoint rule set evaluated in parallel fashion by software agents, hosted on different computing devices, and able to compute appropriate fuzzy operators. In order to fix our transparent FLC view in a multi-agent environment it is necessary to show how FLC tree can be broken in different subtasks, each one representing a portion of initial fuzzy rule base. Let $\Sigma_{RB} = \cup_i E_{FR_i} \cup \{(n_{RB}, n_{FR_1}), (n_{RB}, n_{FR_2}), \dots\}$, be the rule base

of our fuzzy controller and n the number of rules. For instance, this rule set can be simply splitted into two different rule base as follows:

$$\Sigma_{RB}^1 = \{ E_{FR_i} \cup \{ (r_{RB}, r_{FR_{i_1}}), (r_{RB}, r_{FR_{i_2}}), \dots \}$$

$$\Sigma_{RB}^2 = \{ E_{FR_i} \cup \{ (r_{RB}, r_{FR_{i_1}}), (r_{RB}, r_{FR_{i_2}}), \dots \}$$

This splitting produces two different controllers each one dealing with half of rules contained into Σ . The proposed schema allows to create a fuzzy agency characterized by following agents categories: Splitter agents and Fuzzy agents. The *Splitter agent* is devoted to create a collection of disjoint rule set by following the previous schema; the *Fuzzy agent* is a software entity able to compute the fuzzy inference operator on a rule partition coming from Splitter agent. More in details, if n is the cardinality of initial rule base and m is the number of fuzzy agents populating the agency then the splitter agent move at most $\lfloor n/m \rfloor$ rules on each of fuzzy agents.

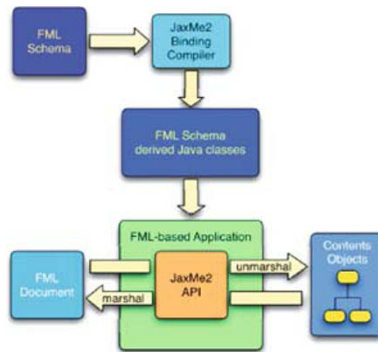


Fig. 3. JaxMe2/FML/Java binding

3.3 Implementing the FML Fuzzy Agents

The FML codes represent only an human-oriented and hardware-independent representation of a fuzzy controller, i.e., the FML programs cannot be computed in a direct way. JaxMe2 represents a direct way to compile and compute the FML services. In fact, the JaxMe2 allows to translate the XML tree structure (in our case, the FOM) into a Java classes hierarchy in direct and simple way through the JaxMe2 compiler. Specifically, JaxMe2 can generate Java classes from XML schemas by means of a JaxMe2 binding compiler. The JaxMe2 binding compiler takes XML schemas as input, and then generates a package of Java classes and interfaces, which reflect the rules defined in the source schema. The JaxMe2 binding framework provides methods for unmarshalling XML instance documents into Java content trees, a hierarchy of Java data objects that represent the source XML data, and for marshalling Java content trees back into XML instance documents. The

JaxMe2/FML/Java binding is depicted in figure 3. In order to complete the Java representation of FML fuzzy controllers, a fuzzy wrapper class, named FML-Controller has been coded. In particular, FMLController class exhibits a set of methods able to apply the appropriate fuzzy operators to the information derived from JaxMe2 objects. In particular, FMLController constructors allow to create a new fuzzy controller by using the unmarshall method of JaxMe2-API independently from FML file location (file system or network).

4 Integrated Development Environment for Transparent Fuzzy Agent Design

The tree representation of a FLC and, consequently, its mapping in FML language offers an additional important benefit: it allows to design and implement a fuzzy controller by means of simple visual steps. In other words, the FLC tree nature allow the designer to draw a controller through drag and drop actions, i.e., at the same way of visual programming languages allow to write computer programs only making use of the mouse device. By exploiting this benefit an advanced integrated development environment for FLC design and implementation has been realized Its main features are: 1) Fuzzy controller drawing; 2) FML modeling, 3) FML controller synthesis. Fuzzy controller drawing and FML modeling have been discussed previously. FML controller synthesis is a further benefit coming from XML nature of Fuzzy Markup Language. In fact, proposed application exploits the JaxMe2 tools in order to translate the FML abstract controller view in a real implementation based on Java computer languages. Figure 5 shows a screenshot of realized IDE application (FuzzyIDE). Through proposed approach, the MAS developer can define the agent intelligence simply drawing a fuzzy tree and, successively, they can 'compile' this tree obtaining an computable object.

5 Conclusion

In this paper, several principles for structuring systems support for smart multi-agent environments have been presented. Precisely, a novel model for fuzzy controller has been realized; this new representation, based on labeled tree structure, allows to design transparent fuzzy controller, i.e., control systems able to be reprogrammed on different hardware without repeating the design and implementation steps. Moreover, the recursive nature of labeled tree allows to embed the transparent fuzzy controllers in a distributed environment (multi-agent system) in a direct way. The joint use of labeled tree and multi-agent environment has allowed the implementation of an advanced graphical environment allowing the fuzzy designers to draw their ideas by means of distributed fuzzy controllers achieving a drastic decrease of control systems development time.

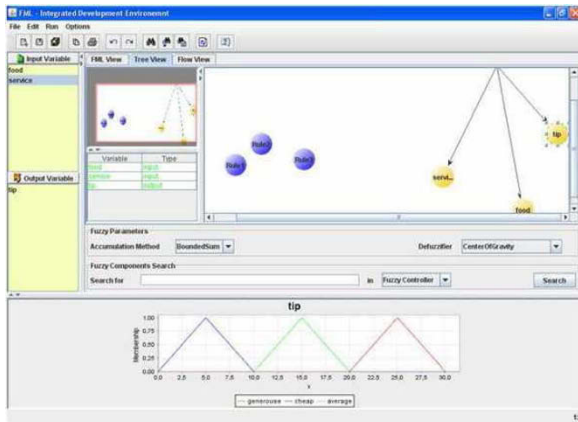


Fig. 4. FML IDE Screenshot

References

1. Ferber J (1999), *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, Addison Wesley.
2. Acampora G, Loia V (2005) Fuzzy control interoperability and scalability for adaptive domotic framework, *IEEE Transactions on Industrial Informatics*, Vol. 1, No. 2, pp. 97–111.
3. Mamdani EH (1974) Applications of fuzzy algorithms for simple dynamic plants, in *Proc. IEE*, vol. 121, pp. 1585–1588.
4. Zadeh LA (1965) Fuzzy set, *Inform. Control* vol. 8, pp. 338–353, 1965.
5. Hagra H (2007) Type-2 FLCs: A New Generation of Fuzzy Controllers, *IEEE Computational Intelligence Magazine*, Vol. 2, pp. 30–44.
6. Wang YL, Chen HC, Liu WK (1996) A Parallel Algorithm for Constructing a Labeled Tree, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, pp. 1236–1240.
7. Acampora G, Loia V (2004) Fuzzy control interoperability for adaptive domotic framework, *IEEE International Conference on Industrial Informatics*, pp. 184–189
8. Acampora G, Loia V (2006) Ubiquitous Fuzzy Computing in Open Ambient Intelligence Environments, *IEEE International Conference on Fuzzy Systems*, Vancouver, Canada, pp. 465–470

Archetypal Internet-Scale Source Code Searching

Medha Umarji¹, Susan Elliott Sim², and Crista Lopes²

¹ University of Maryland Baltimore County,
Department of Information Systems
Baltimore, Maryland, USA
medhal@umbc.edu,

WWW home page: <http://userpages.umbc.edu/~medhal/>

² University of California, Irvine, Department of Informatics,
Irvine, California, USA
{ses, lopes}@ics.uci.edu,

WWW home page: <http://www.ics.uci.edu/~ses/>

WWW home page: <http://www.ics.uci.edu/~lopes/>

Abstract. Programmers often search for Open Source code to use in their projects. To understand how and why programmers search for source code, we conducted a web-based survey and collected data from 69 respondents, including 58 specific examples of searches. Analyzing these anecdotes, we found that they could be categorized along two orthogonal dimensions: motivation (reuse vs. reference example) and size of search target. The targets of these searches could range in size from a block (a few lines of code) to a subsystem (e.g. library or API), to an entire system. Within these six combinations of motivations and target sizes, nine repeating motifs, or archetypes, were created to characterize Internet-scale source code searching. Tools used for searching and the criteria for selecting a component are also discussed. We conclude with guidance on how these archetypes can inform better evaluation of Internet-scale code search engines, as well as the design of new features for these tools.

Keywords: *Source code, search engine, empirical study, Open Source*

1 Introduction

The growth of Open Source has made a “rich base of reusable software” available on the Internet that programmers leverage in their own software development projects [1]. As this opportunistic software development becomes increasingly prevalent, finding the right snippet, component or application to reuse becomes a common activity for developers. However, little is known about how they go about these searches. To this end, we conducted an online survey of programmers and asked them to give us specific examples of how they searched for source code, the tools they used, and criteria for making a final selection.

Please use the following format when citing this chapter:

Umarji, M., Sim, S.E. and Lopes, C., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 257–263.

A total of 69 participants responded to our survey and contributed 58 anecdotes about how they searched. By analyzing the anecdotes qualitatively and inductively, a number of patterns or “archetypes” were identified. These anecdotes could be categorized along two orthogonal dimensions: motivation and size of search target. With respect to motivation, participants tended to be looking for code that could be reused with little or no modification, or they were looking for code to use as a reference example. Often a search began by looking for code to reuse without modification, but when one was not found, a close match was used as a reference example. Within these categories, we identified nine different archetypes.

The remainder of this paper is organized as follows. We describe our research method in Section 2. In Section 3, we present our categorization and archetypes. In Section 4, we discuss how our participants conducted their searches. We explore how these archetypes can be applied in Section 5 and we make some concluding remarks in Section 6.

2 Online Survey

Surveys have become an established empirical method, especially for Internet usage [2][3]. We designed an online survey with 13 closed-ended questions and two open-ended questions. The survey also had questions about the different tools used during search, programmer background, and criteria for finally selecting a component. In this paper, we will be focusing on the data from one of the open-ended questions, which asked:

Please describe one or two scenarios when you were looking for source code on the Internet.

(Please address details such as: What were you trying to find? How did you formulate your queries? What information sources did you use to find the source code? Which implementation language were you working with? What criteria did you use to decide on the best match?)

We solicited responses using convenience sampling by sending invitations to participate in the survey to the following newsgroups and mailing lists: Javaworld mailing list, the CGI beginner’s list on perl.org, comp.software-eng, comp.lang.c, and comp.lang.java. While our responses give us a broad overview of how software developers search for source code, there is no way to determine whether we obtained a representative, random sample. Consequently, we present our results as qualitative archetypes rather than quantitative statistics.

An archetype is a concept from literary theory. It serves to unify recurring images across literary works with a similar structure. In the context of source code searching, an archetype is a theory to unify and integrate typical or recurring searches. The archetypes were produced by analyzing the anecdotes for recurring patterns using open coding [4] and a grounded theory approach [5].

3 Archetypes

In this section, we present the archetypes that we found in our study. Each theme or archetype was a combination of a search motivation and size of search target with additional details about the context.

Detailed analysis of scenarios showed that respondents were either searching for *reusable code* (34) or *reference examples* (17). A key difference between the two motivations is how much modification was expected to reuse the code. Reusable code is source code that they could just drop into their program and use right away, such as a component implementing a quick sort algorithm. A reference example is a piece of code that illustrates a particular solution but needs to be re-implemented or significantly modified; for instance, the syntax of a particular method call in Java.

Across both types of searches, the size of the search target varied in a similar fashion. The size was a continuous variable ranging from a few lines to an entire system, but we defined three broad categories in our analysis. The sizes of the search targets were classified as a block (11), a subsystem (32), or a system (8).

Table 1: Motivation by Target Size

	Description	Code for Reuse	Reference Example	Row Total
Block	Wrappers, parsers, code excerpts	7	4	11
Subsystem	Code of algorithms and data structures, GUI widgets, Library, APIs	21	11	32
System	Stand-alone Applications	6	2	8
Column Total		34	17	51

3.1 Motivation: Reuse

As can be seen in Table 1, there were 34 searches that were motivated by reuse without modification. Upon performing a thematic appraisal, we were able to identify four archetypes or themes of searches within the reuse motivation. The archetypes are presented in increasing order of target size.

Looking for code snippets, wrappers or parsers: The seven searches where developers were looking for a block of code to reuse included searches for a “Java wrapper code for the native pcap library” and an “RSS feed parser.” Also included were searches that performed some small functionality, like “encode/decode a URL” and “convert uploaded images of all types to jpeg”. At this level of granularity we

included searches for source code components that solved a small problem, or for code snippets to add functionality (with little or no modification).

Reusable data structures, algorithms and GUI widgets to be incorporated into an implementation: Participants were looking for subsystems such as algorithms and data structures; “two-way hash tables,” “B+ trees,” “trie trees,” and “binary search algorithm” were some of the eight searches included in this archetype. These searches are popular, as there is a close match between the vocabulary in the code and the vocabulary for describing the search. There were also seven searches for graphical user interface (GUI) widgets in our data, for example “inserting a browser in a Java Swing frame.” GUI widgets are easy to reuse, and provided the widget was written using the same GUI framework as the project being developed, there could be relatively few compatibility problems.

A reusable library to be incorporated into an implementation: There were six searches for a reusable package or API. Some examples of the searches were for “speech processing toolkits”, “library for date manipulation in Perl” and “Java implementations of statistical techniques.”

A reusable system to be used as a starting point for an implementation: In six searches, developers were looking for “stand-alone tools” or a “backbone for an upcoming project” or just a “big piece of code that does more or less what I want.” Examples of search targets included an application server, an ERP package or a database management system. We conjecture that this type of search is common, because a system does not need to be de-contextualized before it is used in a new project. Also, systems are easy to find because they typically have web sites or project pages that contain text descriptions of the software.

3.2 Motivation: Reference Examples

There were 17 searches for reference examples, as shown in Table 1. We also used thematic appraisal on these searches, and coincidentally found four archetypes.

A block of code to be used as an example: Programmers were not looking for reusable components, but their goal was to learn through examples, such as “examples of thread implementation in python.” Respondents reported that “reference examples of language syntax and idioms” were helpful when working with an unfamiliar programming language, and “...mostly I look for code for syntax, I don’t always like to refer to books for syntax if it is readily available on my desktop.”

Example of how to implement a data structure, algorithm or GUI widget: In three instances, developers looked for source code snippets to verify their solution or to aid in re-implementation, e.g. “to verify the implementation of a well-known algorithm.” An implementation is more informative than a description or pseudocode, and a running program has a lot of credibility. There were also two

searches for examples of how to use GUI widgets. These included searches for code samples on how to use a particular component from Swing and Microsoft Foundation Classes.

Example of how to use a library: Developers looked for examples of how to use a library in six instances, such as, “Sometimes; I did a source code searching when I don’t know how to use a class or a library.” Libraries and APIs can be complex to use, and a reference example is easier to understand than documentation, because it gives the programmer a starting point that can then be tweaked to suit the situation.

Looking at similar systems for ideas: While developing or modifying a system programmers look at existing similar systems for ideas. Two searches were for systems from which the logic/principles can be borrowed to construct new systems. This technique was mainly used when it was easier to construct a new system rather than adopt an existing one, or on a proprietary Closed Source project that could not be contaminated with Open Source code.

3.3 Information about Defects

There were five searches that did not fit the above motivations and search target matrix. These were searches for information on resolving defects.

Confirmation and resolution of defects: Developers prefer to search for a patch or quick-fix by forming natural language queries with the keywords from an error message or keywords based on the functionality deviation caused by a bug or defect. There were five searches that were looking for solutions to a defect. Sometimes the solution was in the form of a report or post to a forum that confirmed the presence of a bug. At other times, there was a patch that repaired the defect. Three of the searches led developers to find relevant information in mailing lists and forums.

4 Search Tools

A majority of the developers that responded to our survey were programmers in Java (54), C++ (58) and Perl (32). More than half of the respondents (37) had contributed to an Open Source project. Most of the respondents (41) had experience working on small teams with 1 to 5 people.

Sixty of the participants reported turning to a general-purpose search engine, such as Google or Yahoo! to search for source code. Project hosting sites came next, with 34 of the respondents using them for source code search. Only 11 out of 69 respondents reported using a code-specific search engine. Others were highly skeptical, with one participant writing, “In short, I would never rely on a ‘code search engine’. This idea is just plain silly...sort of ivory tower. If you want to find something usable you have to look for people already using it.” Some did just this, by using social tagging sites, such as (del.icio.us) to find code

While selecting which component to reuse, the most important single criterion was functionality. The type of software license came in second. Help from a local expert, an electronic forum, a mailing list archive, or active users, were other major considerations while choosing Open Source software. One respondent said he looked for "... issues which are then found by people, solved and posted on mailing lists of discussion forums." Overall, social characteristics of the project, such as the level of activity, seem to have precedence over characteristics of the source code, such as whether the code is peer reviewed and tested.

5 Applications of Archetypes

The search archetypes described here are a succinct representation of a developer's search for source code. We intend our results to inform further enhancement and design of these source code search engines, such as Koders, Google Code Search, and Krugle.

Inform the evaluation of code search engines: The archetypes can be instantiated into "scenarios" that closely represent the user's situation and are easily understandable by users. These scenarios can then be used to compare and evaluate the effectiveness of different search engines. For example, a generic archetype is a "Search for a reusable data structure." Now this can be instantiated into a scenario such as "Consider that you are looking for the Trie tree data structure written in C to use in your project". Such a scenario will provide the necessary contextual information for users to judge whether the results returned by search engine A are better or worse than those from search engine B.

Inform the design of additional features for these search engines: Our archetypes also point the way to possible new features in Internet-scale code search engines. For example, we found that users search for *information about a particular bug* within an open source project. This kind of search is currently not well supported by code search sites. Although a bug may not be resolved completely, people on forums and mailing lists may have discussed it, and users have to perform a text-based search to find this information. Another possible feature is *providing usage examples* for a library or package. In case a user searches for a particular library, package or API, his search experience will be enhanced if results also contain links to usage examples. Finally, based on our survey findings we observe that a *user rating or recommendation for each item returned* by a search engine will be helpful to a developer trying to choose between available options.

6 Conclusions

The main contribution of this work is insight into the varieties of Internet-scale source code searches that programmers perform. Until now, there has been little research about what types of source code programmers look for and how they find and use it. We conducted a web-based survey of how people look for code on the Internet. We identified a categorization for searches along two dimensions (motivation and size) and presented nine archetypes of searches within these categories.

The results of this study are applied to the design of tool features, and in the formation of scenarios for evaluating the existing search tools. Tools for locating source code would be complemented by recommender systems and reviews; more like Amazon.com and less like a library catalog. One of the future directions for this line of research is the use of scenarios to evaluate different search engines. We are continuing with experiments in this vein and hope to continue making contributions to Internet-scale source code searching.

References

- [1] D. Spinellis and C. Szyperski, "Guest editors' introduction: How is open source affecting software development?" *IEEE Software* vol. 21 pp. 28-33 2004
- [2] V. M. Sue and L. A. Ritter, *Conducting online surveys*: Sage Publications, 2007.
- [3] S. E. Sim, C. L. A. Clarke, and R. C. Holt, "Archetypal source code searches: A survey of software developers and maintainers " in *Proceedings of the 6th International Workshop on Program Comprehension* IEEE Computer Society, 1998 pp. 180
- [4] M. B. Miles and A. M. Huberman, *Qualitative data analysis*: Sage Publication, 1994.
- [5] A. Strauss and J. Corbin, *Basics of qualitative research: Grounded theory procedures and technique*. Thousand Oaks: Sage Publications, 1990.
- [6] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, C. Lopes. *Sourcerer: A Search Engine for Open Source Code Supporting Structure-based Search*. October 2006. Companion to the 21st ACM SIGPLAN conference on Object-oriented programming languages, systems, and applications OOPSLA '06. ACM Press.

Channeling Firefox Developers: Mom and Dad Aren't Happy Yet*

Jean-Michel Dalle¹, Matthijs den Besten², and H ela Masmoudi¹

¹ Universit e Pierre et Marie Curie, Paris, France;

`jean-michel.dalle@upmc.fr,masmoudi_hela@yahoo.fr`

² University of Oxford, Oxford, UK; `matthijs.denbesten@oerc.ox.ac.uk`

Summary. Firefox, a browser targeted at mainstream users, has been one of the big successes of open source development in recent years. That Firefox succeeded where earlier attempts failed is undoubtedly due to the particular choices that were made in the process of development. In this paper, we look at this process in more detail. Mining bug reports and feature requests related to Firefox in Mozilla's Bugzilla bug tracker system, we find that the attention developers devoted to reports and requests was influenced by several factors. Most importantly, other things being equal, reports and requests from outsiders increasingly tend to be ignored. While such behavior may have helped to shield Firefox from the "alpha-geek power user" in the early stages of development, it also makes it difficult for "mom and dad" to let their voice be heard even after they have adopted Firefox.

1 Introduction

In June 2006, Blake Ross, one of the initiators of Firefox, gave an interview to Olivia Ryan. At some point, the conversation turned to the issue of project management [5]:

Olivia Ryan: And so was it difficult sometimes to strike that balance between working on an open project and trying to keep end-users in mind? Blake Ross: Yes. Yes. Everybody hated us for a long time.

Everybody may have hated Blake Ross and his companions, but they succeeded: In 2004 Firefox appeared and it quickly became, as an article in Wired put it, "the hot new browser rocking the software world." What we would like to know now is how they succeeded.

Understanding the processes that helped shape Firefox is important for a variety of people. First of all, as more and more users adopt Firefox as their default browser, they need to be reassured that Firefox remains a stable and secure alternative that is geared towards their needs. But also the stakeholders

* The work presented here has benefited from discussions with Paul A. David and others in the context of the OII/MTI project on Distributed Problem Solving Networks.

Please use the following format when citing this chapter:

Dalle, J.-M., den Besten, M. and Masmoudi, H., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Bj orn Lundell, Giancarlo Succi; (Boston: Springer), pp. 265–271.

of companies that have invested in Firefox, or that consider participation in similar projects, need to ascertain whether the Firefox approach to open source development works. Ultimately, even the developers themselves, who may not always be aware of the processes that channel their behavior, could benefit from our exercise in data mining.

Methodologically, we subscribe to the suggestions of Scacchi and Sack and others [7, 6] that the processes of open source software development can be discovered by studying the patterns of interactions between the developers and the resulting code. Our recurring obsession, in this study and in previous studies, is to identify what one could call *stigmergic* features in the interactions [1]. Like pheromone guiding ants from the nest to the food source, we consider that, lacking strict managerial control, there must be elements in the code and documentation that nudge developers to devote their attention in specific directions. Previous results of ours suggest that developers do indeed react to signals like the complexity of the code and address *complex* tasks in teams [3]. Further, we found that *contextual* elements, be it the level of detail in the specification of the task or linkage with related tasks, affect the speed with which tasks are dealt with and we also found that a lack of contextual elements may make it necessary to revisit the task several times [2]. In the study here, we focus at the attraction of ownership: Does it make a difference whether the task stems from the project *core* or from its *periphery*? In particular, does it make a difference for the resolution of a bug, whether the bug was reported by an insider or by an outsider? Short answer: yes. The long-term research agenda to which this study contributes is to derive an ontology of distributed process management from the structured, semi-structured, and unstructured data that document open source software development. Our hope for our ontology is that it could help assess the sustainability and evolution of projects like Firefox.

“We’re making a product for mom and dad.” Blake Ross said of Firefox [4]. That is probably not the case for our research. Nevertheless, in what follows, we will make a valiant attempt to share our preliminary findings. First, we describe in greater detail the bug-report data that we collected and how we connected them with information on bug-histories and code-patches. Next, we will present our preliminary results which suggest that the same process that helped make Firefox a product for mom and dad in the first place now threaten to lead to a neglect of mom and dad’s needs in the future.

2 Data

Sack and others note that developers coordinate their activity almost exclusively in three information spaces: “the implementation space, the documentation space, and the discussion space” [6]. Our collection covers each of these spaces, at least partially. It covers the implementation space in so far as we only consider bugs that are mentioned in commits to the official code base that were kept by the Mozilla concurrent versions system (CVS). For those bugs, in so far

as they are associated with a file of which at least one revision is part of a Firefox release, we retrieve the bug-reports and the logs of changes to those bug reports kept by the bug tracker system Bugzilla. Thus, we are covering the documentation space as well. We also cover the discussion space since mailing-list style comments relating to the bugs are included in the reports. However, if Blake Ross is right in his assessment that Internet Relay Chat “was a big means of communication and especially with Firefox early on” [5], then we might have a problem, because, to our knowledge, archives of those chats have not been kept.

On the other hand, so far, we haven't done the text mining to which the chat archives and bug comments lend themselves. Instead, for now we have focused on a range of more readily quantifiable indicators. From the CVS, we retrieved for each bug the number of different authors referring to the bug in their commit-comments; the number of files touched by these commits; the number of distinct comments and the number of commit; and, finally, the number of lines of code added and deleted through the commits. From the bug-reports, we retrieved the number of persons copied in the bug resolution discussion; the number of other bugs the bug depends on or blocks; the numbers of attachments, patches and comments and the number of distinct contributors to the discussion; and, besides, the number of bugs that were identified as a duplicate of the bug reported. In addition, we retrieved the priority assigned and the severity estimated for each bug. With help of the change-log of the bug-reports, we also retrieved the time that passed between the opening of the report and the assignment of someone in charge of the resolution process; we retrieved the number of times the bug was (re-)assigned whether its priority was incremented or decremented or had been changed more than once; whether its severity increased or decreased or changed more than once — where the importance was judged to be in order of trivial, enhancement, minor, normal, major, critical, blocker. We determined whether the initial status of the bug report was **New** or **Unconfirmed**; whether the bug was reopened at least once; the number of report edits by the bug-reporter, and the number of report edits by the last person in charge of its resolution.

We drew two samples of bugs from the 40 000 or so that have resulted in a change to the Firefox code base. The first from the early stages of the project, when Firefox was still called Phoenix, in between release 0.1 and 0.5; the second from a later, more mature, phase between release 1.5 and release 2.0. For the moment, we ignore bugs where the severity was judged to be enhancement as we want to focus on bug reports rather than feature requests.

3 Results

Blake Ross described the Firefox development philosophy thus [4]:

We're making a product for mom and dad. You still have a voice here, but some of the features that you think we should add may not be the

ones that they want to use. So you have to take our word for it that, even though 500 of you want something right now, you may actually be in the minority of a much larger group that we’re pursuing that’s going to be silent during this phase of development.

Bug reports provide an invaluable window where we should be able to see this philosophy in action. It is there that we can witness the interaction between users and developers and it is also there that we can see how developers manage to balance the demands from the early adopters, the “alpha-geek power user” against those from those targeted, mom and dad. Take for instance bug 213186². This bug report is a request to alter the text in the preference pane and in particular to replace the geek-humour of “Cookies are delicious delicacies” with a more appropriate explanation for concerned users without prior knowledge of the concept. Blake Ross himself was responsible for the “delicious delicacies” joke in the first place and his mom and dad might well have approved. Many other users however might be put off by the lack of information provided by people who seriously want to consider whether want to accept cookies or not. And the eventual resolution of bug 213186 shows that the concerns of these people were eventually recognized as “delicious delicacies” was replaced by “pieces of information stored by web pages on your computer[...].” Table 1 gives a few indicators for the extent and the duration of the bug-report. From these indicators, it is already clear that considerable effort was required to resolve this bug — effort that was mainly devoted to the construction of consensus between the geeks who like this humour and the advocates of mom and dad who did not.

Table 1. Characterization of bug 213186—*Please remove ‘Cookies are delicious delicacies’ from Options→Privacy→Cookies* and bug 171349—*Mozilla Firefox Icon is Windowing System’s Standard Icon*.

Parameter Description	Bug 213186	Bug 171349
Reported	2003-07-19	2002-09-28
Fixed	2004-12-03	2005-05-30
Initial Status	Unconfirmed	New
Severity	trivial	normal
Time to First Assignment(<i>hrs</i>)	314	576
Number of Comments	55	206
Number of Authors of Comments	18	89
Number of Duplicates	1	31
Number of Patches	9	11

Bug 171349³ is the other bug that is characterized in the table. The subject matter is less exciting — a lot of the attention for bug 213186 could be because

² https://bugzilla.mozilla.org/show_bug.cgi?id=213186

³ https://bugzilla.mozilla.org/show_bug.cgi?id=171349

it is a bike shed like discussion in which it is easy to have an opinion⁴ — but that does not make this bug less important *per se*. What is interesting about this bug is why it took so long to resolve. As one commenter remarks:

> This bug was first discovered about 2 1/2 years ago and is still listed as “NEW”?

Mozilla, like all the other big projects, has a LOT of bureuacracy [sic] going on. Ireported this so long ago I forgot all about it, and probably would've submitted it again if it weren't for the recent activity of my report being marked as a dupe... But even if it requires 5 people to check it and give it their OK, surely adding a .ico to the package can't be /that/ hard, can it? (Shish, 16/03/2005, comment #165)

As bug 171349 is the kind of bug that typically matters more to mom and dad than to a power user, the question is relevant whether the difficulty of the resolution of this bug was typical in Firefox development or the exception.

In order to get an impression of the general patterns of bug resolution in Firefox, we performed a survival analysis of a variety of samples of bug-reports related to Firefox. Table 2 presents the results. We carried out regressions on six samples: A sample of bugs related to Firefox in the early phases of its development; a sample of bugs related to Firefox in a later phase of its development; and for both of these samples a sub-sample of bugs started their life as *New* and another sub-sample for those bugs that started their life as *Unconfirmed*.

The difference between *New* and *Unconfirmed* is due to the fact that only a subset of the members of the Firefox community have the so-called *CanConfirm* privilege, according to which their bugs start as *New* and do not need confirmation — which is precisely needed otherwise to move from *Unconfirmed* to *New*. This privilege is granted by cooptation, based on the past track record of developers asking for it, and we therefore consider for the sake of the analysis here that it characterizes members of the core of the community, as compared to more peripheral members without this privilege.

What emerges from the table is that bug treatment in the early phases of Firefox was different from treatment at a later stage. More in particular, the difference between the treatment of bugs that stem from outsiders in contrast to the treatment of bugs in general seems more pronounced in the more recent samples. Besides, issues like the complexity of the problem and effort devoted to the contextualisation of the problem still play a role in determining the speed with which a bug is typically resolved.

4 Conclusion

Firefox is an open source project that so far has enjoyed a wide appeal among mainstream users as well as developers. In this paper, we argued that a study

⁴ see <http://www.bikeshed.org>

Table 2. Significance and impact of variables controlling for bug fixing regimes (Survival analysis; Weibull fit).

Parameter Description	Sample: Phoenix (< 0.5)			Firefox (> 1.5)		
	All	New	Unconf'ed	All	New	Unconf.
# Bugs	7596	6200	1366	4721	3465	1256
Intercept	***	+	**	***	+	**
Number Of Different Committers	***	***	***	***	***	
Number Of Files Touched in Codebase	***	***	***	***	**	•
Number Of Different Comments in Commits				***	***	
Number Of Commits	-*		***	***	-*	
Number Of Lines Of Code Added	***	***	***			
Number Of Lines Of Code Deleted			+			
Number Of Persons Copied	**	**			+	***
Number Of Attachments	+		+			
Number Of Patches				•		
Depends On How Many Other Bugs	**	+	+			•
Blocks How Many Other Bugs	+	**				
Number Of Comments			•			**
Number Of Authors Of Comments	***	***	***	***	***	***
Severity Trivial	***	***	***	***	***	
Severity Minor	***	***	***	***	***	
Severity Normal	***	***	***	***	***	
Severity Major	***	***	***		+	
Severity Critical	***	***	***		**	-*
No Priority	-*	-*				
Priority 1	**	-*	-*			
Priority 2	-*					+
Priority 3	-*	-*				•
Priority 4						
Log Time To First Assignment	***	***	***	***	***	***
Number Of Times Bug Was Assigned	***	***	***	***	***	***
Priority Was Increased	+	•				
Priority Was Decreased	**	+	•	**	**	
Priority Was Changed More Than Once	•	•				•
Severity Was Increased				**	***	
Severity Was Decreased				+		+
Severity Was Changed More Than Once				**		
Initial Status Was New						•
Bug Was Reopened At Least Once	•					-*
Number Of Edits By Bug Reporter			•	**	**	**
Number Of Edits By Last Assignee		***	•		+	
Number Of Duplicates		-*	-*	***	***	

of the Firefox bug resolution processes is crucial to an understanding of the dynamics of its development. We described bug-reports and their association with code-commits. We described two bugs in detail and we discuss our preliminary efforts to detect patterns in the survival of bugs in general. Our findings suggest that Firefox developers tend to ignore the concerns that are voiced by outsiders, especially so in the later stages of its development. Does this mean that mom and dad, for whom Firefox was meant, are increasingly ignored as well — that remains to be seen. If this were true, however, it wouldn't bode well for Firefox' future.

The research described here constitutes just the first steps of a more ambitious research agenda in which the ultimate goal is to derive an ontology of distributed project management. What we have done here is to investigate the interaction between core and periphery in one project. For now, all we have is a suggestion that the current situation might not be optimal. But as our understanding grows and as our ontology expands, we could imagine a future in which the ontology itself would be used to optimize the development processes of open source software by accounting for the management of attention. And that could be a further step in making mom and dad happier.

References

1. J.-M. Dalle and P. David. Simulating code growth in libre (open-source) mode. In N. Curien and E. Brousseau, editors, *The Economics of the Internet*. Cambridge University Press, Cambridge, UK, 2007.
2. J.-M. Dalle and M. den Besten. Different bug fixing regimes? A preliminary case for superbugs. In *Proceedings of the Third International Conference on Open Source Systems*, Limerick, Ireland, June 2007. To appear.
3. M. den Besten, J.-M. Dalle, and F. Galia. Collaborative maintenance in large open-source projects. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, and G. Succi, editors, *Open Source Systems*, volume 203 of *IFIP International Federation for Information Processing*, pages 233–244, Boston, 2006. Springer. Received best paper award.
4. J. Livingston. Blake Ross; creator, Firefox. In *Founders at Work: Stories of Startups' Early Days*. Apress, 2007.
5. O. Ryan. Interview with Blake Ross. Archived at <http://mozillamemory.org>, June 2006.
6. W. Sack, F. Détienne, N. Duchenaud, J.-M. Burkhardt, D. Mahedran, and F. Barcellini. A methodological framework for socio-cognitive analyses of collaborative design of open source software. *Computer Supported Cooperative Work*, 15(2-3):229–250, 2006.
7. W. Scacchi. Socio-technical interaction networks in free/open source software development processes. In S. T. Acuna and N. Juristo, editors, *Software Process Modeling*, pages 1–27. Springer, New York, NY, 2005.

Continuous Integration in Open Source Software Development

Amit Deshpande,

TechSolve, Inc.
6705 Steger Drive
45237 Cincinnati, OH, U.S.A.

amit@amit-deshpande.com

Dirk Riehle

SAP Research, SAP Labs LLC
3475 Deer Creek Rd
94304 Palo Alto, CA, U.S.A.

dirk@riehle.org

Abstract. Commercial software firms are increasingly using and contributing to open source software. Thus, they need to understand and work with open source software development processes. This paper investigates whether the practice of continuous integration of agile software development methods has had an impact on open source software projects. Using fine-granular data from more than 5000 active open source software projects we analyze the size of code contributions over a project's life-span. Code contribution size has stayed flat. We interpret this to mean that open source software development has not changed its code integration practices. In particular, within the limits of this study, we claim that the practice of continuous integration has not yet significantly influenced the behavior of open source software developers.

1 Introduction

Open source software is having a major impact on software and its production processes. Many software products today contain at least some open source software components. Some commercial products are completely open source software. For commercial software firms it is therefore important to understand open source software development processes and ensure that employed developers are capable of participating in them [13].

Open source software development processes are different from commercial software development processes like the spiral model [5] or agile methods [4]. Open source processes can vary from project to project. With no single well-defined open source process, it is unclear for hiring managers what skills to look for when hiring people for open source software projects.

One common hypothesis is that open source software processes are similar to agile development processes and hence that agile method skills could serve as a proxy for open source software development skills [12] [14]. However, this hypothesis has never been validated. Indeed, it is impossible to prove as long as open source processes remain as vaguely defined as they are today.

This paper analyses the use of continuous integration within open source software projects. Continuous integration is a practice where software developers work

Please use the following format when citing this chapter:

Deshpande, A. and Riehle, D., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 273–280.

in small increments, contributing code to the project frequently, and ensuring that the project compiles and passes its test suites at any time [11].

Continuous integration is a core agile methods practice. If it is also an important open source development practice, managers can hire more confidently developers for open source projects who have been educated in agile methods and who have appropriate experience with continuous integration.

To answer the question “Have open source projects increasingly been using continuous integration?” we looked at the individual actions of developers from 5122 active open source software projects. Specifically, we looked at the size (in source lines of code) of code contributions and their frequency.

The paper is organized as follows. Section 2 frames our hypothesis. Section 3 discusses our database and the details of the validation approach. Section 4 presents the results of the analysis. Section 5 discusses the shortcomings and limitations of the analysis. Section 6 discusses related work. Section 7 concludes the paper.

2 Is Open Source Using Continuous Integration?

Agile software development methods like Extreme Programming “embrace change”: There is no insistence on following a master-plan, but the willingness to respond to (requirements) change quickly [3]. This is in stark contrast to traditional approaches to software development with their insistence on detailed planning ahead.

It is not obvious how open source projects relate to agile methods. According to Fugetta, “[Agile methods] can be equally applied to proprietary and open source software” [1]. Thus, with the advent of agile methods around 1997/1998, one might expect to see an adoption of agile methods practices in open source software development over the last ten years.

Thus, we are asking: *What is the impact of (the arrival of) agile methods on open source software development processes?* And then, more specifically: *Is continuous integration a widely used practice in open source software projects?* The answer to this question is important to software firms who engage with open source software projects, because they need to hire, cultivate, and maintain the appropriate skill sets of their employees to work with open source projects.

We chose continuous integration as a representative practice of agile methods like Extreme Programming [3] or SCRUM [6]. “Continuous integration is the practice of making small well-defined changes to a project’s code base and getting immediate feedback to see whether the test suites still pass” [11].

If continuous integration had not been employed in open source projects before its first formulations around 1997 but has been increasingly employed since then, we would expect to see a statistically significant behavioral change in how open source software developers contribute code to open source projects. Specifically: *We would expect to see that the average size of code contributions, the individual check-in into a source code repository, would have gone down over the last ten years, and we would expect to see that the average frequency of such check-ins has gone up.*

To investigate this hypothesis, we analyzed more than 5000 active open source software projects, as discussed in the following sections.

3 Data Source and Approach

For the analysis, we use the database of the open source analytics firm Ohloh.net [10]. The database contains 5122 current and active open source projects from 1990 until May 2007.

We analyze the last 16 years from January 1990 to December 2006. The database contains the most popular open source projects as measured by the number of in-links to their website. The in-links are provided by the Yahoo! search engine.

Ohloh.net goes down to the level of the individual developer action. Specifically, Ohloh.net provides each individual commit action of all the projects over their entire history. A commit is the action with which a developer contributes a piece of code to the project's repository.

3.1 The Commit Action

A commit action is the atomic contribution of source code to a code repository. We measure the amount of work that went into a commit in Source Lines of Code (SLoC) added, changed, and removed, ignoring empty lines and comments. SLoC is based on applying the Unix diff command to two consecutive source file versions.

Unfortunately, Ohloh.net counts a changed line as an old line removed and a new line added. One line of code edited will be tracked as one line added and one line removed. Table 1 represents the resulting three cases. From this data alone we cannot determine the exact number of lines of code affected. Thus, in Section 3.3 we will take two approaches representing the upper and the lower bound respectively.

3.2 Commit Action Filtering

We want to study the size of a commit over time. In order to perform a valid analysis we impose four filters to exclude special and undesired events:

1. The initial commit (creation) of a file
2. The final commit (deletion) of a file
3. Commits of a size larger than three standard deviations above the average
4. Commits of a size smaller than three standard deviations below the average

Filter 3 eliminates the cases where a developer adds a large amount of external code to an existing file and Filter 4 eliminates commits where a large amount of code is removed, like in a major refactoring. None of these events helps us understand changes in commit size patterns.

Table 1: Data semantics (SLoC = Line of Source Code)

Case	Commit Action	SLoC Added	SLoC Removed
1	One line of code added	1	0
2	One line of code removed	0	1
3	One line of code edited	1	1

3.3 Analysis of Commit Size

Approach 1 adds the number of lines of code added in one commit to the number of lines of code removed in the same commit. Following the discussion of Section 3.1, editing a single line will be counted as one line added plus one line removed. Thus, the size calculated by Approach 1 represents the upper bound for the commit size.

Figure 1 shows the average size of a commit (as defined by Approach 1) per week from 1990 to 2006. The size is shown on the Y-axis and time is shown on the X-axis. To smooth out fluctuations and to highlight the longer-term trends or cycles we fit a moving average (period = 10) curve on the plot. The commit size is nearly constant and shows no significant trend or pattern over time.

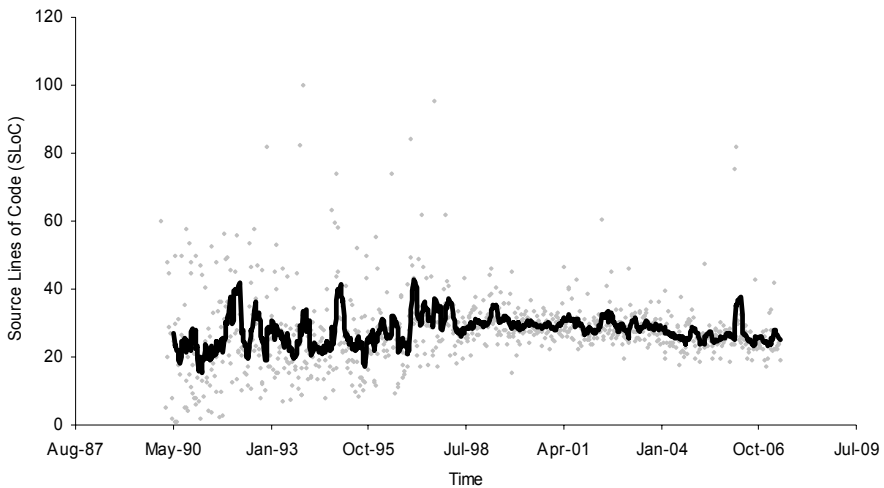


Figure 1: Upper bound of commit size in SLoC over time (Approach 1)

Approach 2 subtracts the number of lines of code removed from the number of lines of code added. Thus, an edited line will not be counted, and the value reflects the net change to the size of a file only. Thus, the commit size as calculated by Approach 2 represents the lower bound for the actual size of the commit.

Figure 2 shows the average size of a commit (as defined by Approach 2) per week using the same axes labeling and moving average curve. The commit size is nearly constant and shows no significant trend or pattern over time.

We tested the validity of our results using hypothesis testing. The null hypothesis is: The average commit size in any year is equal to the average size during the next year. The research hypothesis is: The average commit size in any year is greater than the average commit size during the next year.

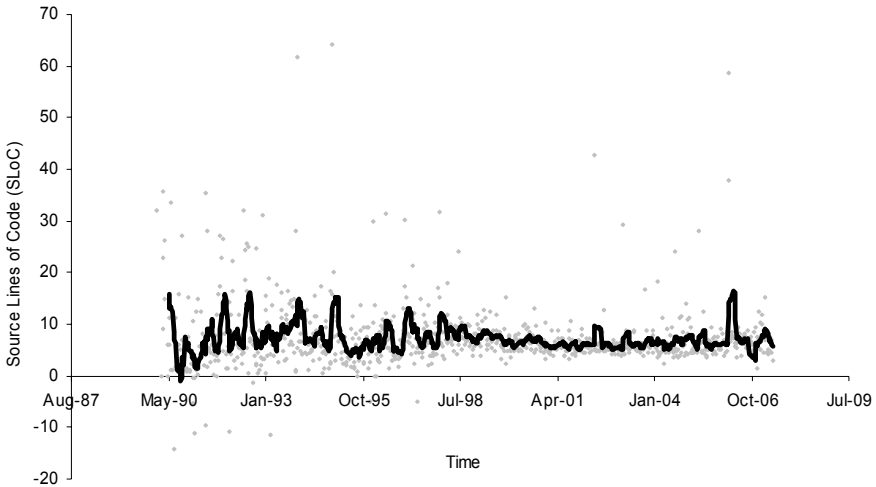


Figure 2: Lower bound of commit size in SLoC (Approach 2)

No clear picture resulted at the 95% confidence level. In a few cases, we could reject the null hypothesis, in most we could not. Within the limits of the study we conclude that the commit size remained constant with no significant trend or pattern over time.

4 Results of Analysis

Our analysis does not validate the hypothesis that open source software developers are practicing continuous integration. Our indicator, the average size of a commit, remained almost constant over the years with no significant trend or pattern. We applied a similar analysis (not given in this short paper) which showed that the commit frequency has remained stable over time as well.

One explanation is that open source projects have not adopted continuous integration on any significant scale. An alternative interpretation is that open source software development has always practiced continuous integration. In that case, the advent of agile methods did not lead to any changes, simply because it had long arrived in open source software development before.

5 Limitations of Analysis

The analysis and the conclusions we draw have at least the following limitations.

- *Sample size.* We considered 5122 open source projects. The total number of open source projects is much larger. However, our data source focuses on active projects of significant popularity. So we believe the sample we are using is relevant for analyzing agile methods practices in open source.
- *Data quality.* We couldn't calculate exact numbers of commit size due to the way Ohloh.net tracks changes to source code files. (See Section 3.1.) We believe that working with a lower/upper bound does not restrict the validity of our results as we care about the trend and not the actual values.
- *Data incompleteness.* Some amount of revision control information in open source projects has been lost forever, as projects have moved from one code repository to another. However, this is old data, and we believe the lack of some of the early histories do not affect the validity of our conclusions.
- *Improper summation across different programming languages.* One may argue that commit size depends on the programming language. We analyzed variation across languages and found that the programming language does not have a significant impact on the results presented in this paper.
- *Improper summation across different project sizes.* Some open source projects are small, some are large. Barry Boehm argues that agile methods don't scale well beyond a certain size limit [2]. Our data confirms within the given limitations that the practice of continuous integration has not found increasing adoption in open source projects over the last ten years.

We are working to remove these and other limitations. However, we believe that while the respective critiques can be made, the effects are limited, as argued above.

6 Related Work

Angioni et al. describe MADD, a “Methodology for Agile Distributed Development” and compare the similarity, differences and applicability of agile practices in open source development [9]. Less than 73% of the developers were knowledgeable about agile development practices. Less than 10% of the developers used core agile practices like pair programming or small iterations.

Turnu et al. studied the effect of applying agile methodology in the open source development environment [8]. Specifically, they studied the effects of applying Test Driven Development on open source development by using a simulation model developed from the Apache HTTP server project data. They concluded that use of agile methods in open source development can yield better results in terms of code quality.

7 Conclusions

Software firms seeking to work with open source software need to understand open source software development processes. Our analysis of the size of code contributions by open source software developers indicates that the core agile methods' practice of continuous integration has not had a significant impact on open source development practices. Hence, we conclude within the limits of our study that agile method skills are not a good proxy for open source software development skills.

Acknowledgements

We would like to thank Prem Devanbu and Gregorio Robles for their feedback on earlier versions of the paper as well as their encouragement for the work presented. We also would like to thank Oliver Arafat and Mario Fernandez for proofreading the paper. Finally, Amit Deshpande performed parts of this work while being employed at SAP Labs, LLC.

References

- [1] Fugetta, A.: Open Source Software—An Evaluation. *The Journal of Systems and Software* (66) 77-90.
- [2] Boehm, B.: Get Ready for Agile Methods, with Care. *IEEE Computer* (January 2002) 64-69.
- [3] Beck, K.: *Extreme Programming Explained: Embrace Change*. Addison Wesley, 1999.
- [4] Cockburn, A.: *Agile Software Development*. Addison-Wesley, 2001.
- [5] Boehm, B.: A Spiral Model of Software Development and Enhancement. *IEEE Computer* (May 1988) 61-72.
- [6] Schwaber, K.: *Agile Software Development with SCRUM*. Prentice-Hall, 2001.
- [7] Lawton, M.: Open Source Business Models. IDC, 2007. Retrieved on Sept 13, 2007, from http://www.idc.com/getdoc.jsp?containerId=IDC_P13018
- [8] Turnu, I., Melis, M., Cau, A., Setzu, A., Concas, G., and Mannarò, K. Modeling and Simulation of Open Source Development using an Agile Practice. *J. Syst. Archit.* 52, 11 (Nov. 2006) 610-618.
- [9] Angioni, M., Sanna, R., Soro, A.: Defining a Distributed Agile Methodology for an Open Source Scenario. In Scotto, M., Succi, G. (eds.): *Proceedings of the First International Conference on Open Source Systems*. Springer Verlag (2005) 209-214.
- [10] Ohloh Corporation, see <http://www.ohloh.net>.
- [11] Duvall, P.: *Continuous Integration: Improving Software Quality and Reducing Reducing Risk*. Addison-Wesley, 2007.

- [12] Koch, S.: Agile Principles and Open Source Software Development: A Theoretical and Empirical Discussion. In: *Extreme Programming and Agile Processes in Software Engineering*. Springer-Verlag (2004) 85–93.
- [13] Morkel Theunissen, WH, Boake, A. Kourie, DG. In Search of the Sweet Spot: Agile Open Collaborative Corporate Software Development. In *Proceedings of the 2005 SAISCIT*: 268-277.
- [14] Warsta, J. and Abrahamsson, P. Is Open Source Software Development Essentially an Agile Method? In: *3rd Workshop on Open Source Software Engineering*, Portland, Oregon, USA.

Extracting Generally Applicable Patterns from Object-Oriented Programs for the Purpose of Test Case Creation

Richard Torkar, Robert Feldt, and Tony Gorschek
Blekinge Institute of Technology,
School of Engineering,
372 25 Ronneby, Sweden
{rto|rfd|tgo}@bth.se

Abstract. This paper presents an experiment performed on three large open source applications. The applications were instrumented automatically with a total of 10,494 instrumentation points. The purpose of the instrumentation was to collect and store data during the execution of each application that later could be analyzed off-line. Data analysis, on the collected data, allowed for the creation of test cases (test data, test fixtures and test evaluators) in addition to finding object message patterns for object-oriented software.

1 Introduction

In order to automate software quality assurance to a high(er) extent, one needs to look at techniques that are suitable for this purpose. In this respect random testing is a likely candidate for automation due to its nature where a minimum of human intervention is needed [1]. Unfortunately, random testing of object-oriented software has not been researched widely (for an overview please see [2]). One of the reasons for this is surely the fact that random testing traditionally compares well to other techniques when it comes to dealing with scalar (pre-defined or primitive) types, while usually is seen to have weaknesses dealing with compound (programmer-defined or higher-level) types. Another reason might be that random testing, again traditionally, has been used on small and delimited example.

This paper aims to:

1. Show how object message pattern analysis, from automatically instrumented applications, can be used to automatically create test cases. (Test cases in this context equal test data, evaluators and fixtures.)
2. Point to how random testing can be performed on these test cases.
3. Examine the possible existence of object message patterns in object-oriented software in addition to the question of general applicability of said patterns.
4. Point out current possibilities for using instrumentation of software items as an underlying foundation to automated software testing.

Please use the following format when citing this chapter:

Torkar, R., Feldt, R. and Gorschek, T., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 281–287.

Throughout this paper an empirically descriptive model is used, i.e. explaining a phenomenon sufficiently on ‘real-life’ software. Next, related work is presented. Following that, the setup of the experiment is presented (Sect. 2) while the results are covered in Sect. 3. Finally, this paper ends with conclusions and future work (Sect. 4).

1.1 Related Work

Related work for this paper can be divided mainly into four categories: random testing, type invariants, dynamic analysis and patterns in object-oriented software.

First, random testing [1], which acts a basis for our approach, is a fairly old research area which has seen several new contributions lately which directly or indirectly affect this paper.

Second, likely invariants has gained considerable attention following the work of Pacheco et al. [3] in addition to Yang et al. [4]. The concept of likely invariants is built upon the assumption that one can find *different* input values, for testing a software item, by looking at the *actual* values the software uses during a test run. Our approach does not focus on likely invariants per se but rather class interactions (not component interfaces [5]), which in its turn provides the order of the methods being executed as well as method signatures and actual input and return values and types.

Third, the concept of dynamic analysis, i.e. analysis of the software based on its execution, can in our approach best be described as a see-all-hear-all strategy. This way it resembles the omniscient debugger [6] that takes snapshots of every change in the state of the running application thus allowing the developer to move backwards and forwards when debugging the application. The difference in our approach is that every important piece of information during execution, is stored for later off-line analysis.

Finally, an overview of object-oriented software testing, and especially patterns in this area, can be found in e.g. [7]. Important to note, in this circumstance, is that the word pattern (as used by almost all of the references) has in many ways a different meaning compared to how it is used in this paper.

In order to clarify the concept of patterns, we use the name *Object Message Patterns* for our purposes. *Object* stands for the fact that the focus is set on object-oriented software. *Message* is short for the message-driven perspective as employed by object-oriented software and finally, *patterns* stands for the execution traces as found when analyzing software.

2 Experimental Setup

In this experiment the focus is set around testing intermediate representations of source code from open source software. Today, in industry, many would say that the centre of attention is mostly around the Java Virtual Machine and the Common

Language Infrastructure (CLI) with its Common Language Runtime (both inheriting from the original ideas brought forward by the UCSD P-System's developers in the late 70's [8]).

The experiment was performed on three different open source software items: Banshee (an open source media player), Beagle (an open source search tool) and the Mono C# compiler, Msc (an open source implementation of the ECMA-334/335 standard). The selection of the software items was performed with the following in mind:

- The application should be written in a language, which can be compiled to an intermediate representation (in this case the Common Intermediate Language).
- The application should be sufficiently large and thus provide a large amount of data for later analysis.
- Separate development teams should have developed the applications.

In the end, Banshee, Beagle and Msc, were considered to fit the profile for the case study. For each application, one use case was selected which would be executed after the application had been instrumented:

- *Banshee*: Start application, select media file, play media file, stop playback, shut down application.
- *Beagle*: Start search daemon in console (background process), perform query in console, close the GUI which presents the search results, stop search daemon.
- *Msc*: Compile a traditional 'Hello World' application.

Table 1. An overview of the experiment showing the number of LOC (lines including comments, white spaces, declarations and macro calls), the size of the assemblies (in KB), the number of classes that were instrumented and the number of instrumentation points (IP), for each application. In addition the time to instrument (TTI) and execute (TTE) with and without instrumentation for each application is presented (in seconds).

App.	LOC	IL (KB)	#Classes	#IP	TTI	TTE w. instr.	TTE w/o instr.
Banshee	53,038	609	414	2,770	28	424	63
Beagle	146,021	1,268	1,045	5,084	71	74	6
Msc	56,196	816	585	2,640	166	34	0.8

After the selection of the candidate applications was accomplished the actual instrumentation took place (see Table 1 for some descriptive statistics). To begin with, each valid class in the Software Under Test (SUT), disregarding abstract, extern and interface annotated signatures, in every assembly (exe and dlls), had instructions inserted in each method which would collect runtime input and return value(s) and type(s), as well as the caller (i.e. what invoked the method). All this data, together with a time stamp, was then stored during runtime in an object

database while the SUT was executed following one of the before mentioned use cases. That is to say, each time a method was executed, during the execution of a use case, an object containing all the values necessary to recreate that state (with its connections), was stored in the object database (i.e. a form of deep copy was saved). Having to serialize the data beforehand would be too resource intensive for obvious reasons not to mention posing some difficulties from a technical perspective.

Next, the content of the object database was analyzed looking for patterns and discovering likely critical regions. The selected paths could then be used for creating a test case (using the actual runtime values as test data and test evaluators). The execution of the use cases, as well as the instrumentation of the assemblies, was performed on Linux 2.6.15, Mac OS X 10.4.4 and Windows XP SP2 using Cecil 0.3 (open source assembly manipulator), AspectDNG 0.47 (open source tool to support aspect oriented programming) and the open source (in-memory) object database engine db4o 5.2.

3 Results

The intention of performing Object Message Pattern Analysis (OMPA) on data in this experiment was to find and generalize patterns that later could be used for testing the SUT. In addition to this the hypothesis is that patterns, if found, could be generally applicable to most, if not all, object-oriented software. Since the analysis was performed manually a limitation on the number of analyzed objects was needed. Thus, 300 objects (in a call sequence) from each application were analyzed from an arbitrary point in the object database (selected by a pseudo-random generator as found on pp. 283–284 in [9]). In the end, eight object message patterns were found during the analysis of the data stored in the object database (Tables 2–3, respectively).

The patterns found are of two different categories. Four patterns belong to, what has by us been defined as object unit patterns. Object unit patterns constitute of a sequence of method invocations on *one* object, i.e. methods in an object has been executed in a certain order. Object trace patterns, on the other hand, are slightly different. They cover the path of execution through *more than* one object.

Object Unit Patterns. Four object unit patterns were found during the OMPA (Table 2); these patterns exercised only one object consistently over time and were found in all three applications (needless to say, the names of the objects and classes differed in all three applications, but the pattern can nevertheless generally be applied on all three applications).

The first pattern, the Vault pattern (Table 2), is a straightforward pattern which is executed by first invoking a constructor then invoking a setter and, finally, multiple times, a getter (before a destructor is invoked). This can be seen as a very rudimentary pattern for storing data in an object which then is fetched by one or many objects, and as such is suitable to always execute in a testing scenario, i.e. data is stored in a simple vault. During the analysis the intermediate representation was

used for examining if a method was defined as a getter or setter (property) by searching for the keyword `.property`. There is of course a possibility that a method is *acting* as getter or setter while not being defined as such, but in this analysis these types of methods are disregarded and an emphasize is put on the proper definition of a getter or setter according to the CIL.

Next, the Storage pattern is an evolved Vault pattern and the combinations of setter and getter invocations can be many (Table 2). Hence, the Storage pattern can be constructed in different ways and a combinatorial approach might be suitable in a testing scenario (compared to the Vault pattern which is very straightforward), i.e. data is stored in storage and the storage has (many) different ways of adding or offering content. The reason for distinguishing between Vault and Storage is that a Vault was common during OMPA and as such should always be used when testing object-oriented software, while Storage, on the other hand, is a more complex pattern (more steps performed) and as such needs additional analysis.

The Worker pattern at first glance looks like bad design. An object gets instantiated, and immediately filled with data. A method is next invoked which manipulates the data, returns the manipulated data and, finally, a destructor is invoked. The reason for this design might be to make sure that the method's implementation is accessible by different objects (extended) since it is declared `public`. If one had opted for a method declared as `private` or even `protected`, which could be invoked when the getter is invoked, then there would be no simple way to reuse the implementation.

Finally, the Cohesion pattern is a pattern that executes one or more methods in one object. It does this without *a priori* setting any values and the order of executing the methods is not always important, i.e. each and every method was found to be (by analyzing objects in the object database) an atomic unit with no dependency on other methods in the class and as such the word cohesion (united whole) was found to be appropriate to use.

Table 2. Different object unit patterns found in all three applications. The first column shows the name selected for the pattern and the second column the actual pattern. Abbreviations used: `ctor` and `~ctor` is short for constructor and destructor respectively, while `setter` and `getter` is a method which sets or gets data stored in the object.

Name	Pattern
Vault	<code>ctor</code> → <code>setter</code> → $1 \dots n$ <code>getter</code> → <code>□ctor</code>
Storage	<code>ctor</code> → <code>setter</code> → $1 \dots n$ <code>getter</code> → $1 \dots n$ <code>setter</code> → ... → <code>□ctor</code>
Worker	<code>ctor</code> → <code>setter</code> → method invocation → <code>□ctor</code>
Cohesion	<code>ctor</code> → $1 \dots n$ method invocation → <code>□ctor</code>

Object Trace Patterns. Looking at the object trace patterns, one can see four patterns that can be generally applicable (see Table 3 on next page); these patterns exercise several objects and constitutes sequences of `object:method` invocations.

A fairly common pattern that seems to come up on a regular basis is the Cascading pattern. This pattern instantiates object after object, which all can be of different or same types. The approach seems to be quite common when object-oriented applications are starting up, but in addition shows up in several phases of an application’s lifetime (from start up to shutdown).

Next, the Storing pattern and the Fetching pattern showed up many times as well. These patterns are directly connected to the object unit Storage and Vault patterns, and as such can be combined in many ways.

The final pattern is the Dispatch pattern. The Dispatch pattern simply invokes one method after another (not a constructor though). In most cases the Dispatch patterns ends up with executing the Storing or Fetching pattern as a final step.

Table 3. Different object trace patterns found in all three applications. Abbreviations used: ctor and ~ctor is short for constructor and destructor respectively, while setter and getter is a method which sets or gets data stored in the object. An alphabetical character in front of the abbreviation, i.e. A:ctor, indicates that a type A object’s constructor is invoked.

Name	Pattern
Cascading	A:ctor → B:ctor → C:ctor → ...
Storing	A:ctor → B:ctor; A:method → B:setter
Fetching	A:method → B:getter
Dispatch	A:method → B:method → C:method → ...

3.1 Test Case Creation

Applying test case creation on the example (and on data in the experiment) is fairly straightforward when all entities needed can be found in the object database, i.e. the following data is available: methods’ name, input type and values, and return type(s) and value(s).

In addition, information regarding the caller can be extracted from the object database by simply examining the caller value in the current method being executed (a value stored in the database) or by examining the time stamp for each stored entity in the database.

The above information provides us with an opportunity to automatically create test cases and, in the end, have a simple smoke test or regression test mechanism, depending on the aim of our testing efforts. In short, the software item is executed, test cases are created from the database, patterns are extracted from the database and the SUT is validated. Obviously, the end goal is to make this a fully automatic process.

4 Conclusions and Future Work

This paper presented eight software object message patterns, for testing object-oriented software. As such it indicates in our opinion that the first steps have been taken on the road to extract generally applicable object message patterns for the purpose of testing object-oriented software. In this case study, the patterns are accompanied with automatically generated test cases whose entities (test data, test fixture and test oracle) are retrieved from a database which stores runtime values that are collected when executing a use case on the SUT. These test cases are then stored and used as a simple regression testing mechanism, hence avoiding manual, error-prone and tedious test case creation.

In the future a replication of this study should be made were the focus would be put on large applications and extensive data collection (i.e. collecting more or all use cases during the system testing phase). In addition, a number of issues with respect to optimization must be solved and the analysis must be performed fully automatically for this approach to be beneficial.

References

1. Hamlet, D.: Random testing. In: Marciniak, J.J. (ed.): Encyclopedia of software engineering. John Wiley & Sons, New York (1994) 970-978
2. Binder, R.V.: Testing object-oriented systems: models, patterns, and tools. Addison-Wesley Longman Publishing Co., Inc. (1999)
3. Pacheco, C., Lahiri, S.K., Ernst, M.D., Ball, T.: Feedback-Directed Random Test Generation. Proceedings of the 29th International Conference on Software Engineering. IEEE Computer Society (2007)
4. Yang, J., Evans, D.: Dynamically inferring temporal properties. Proceedings of the 5th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering. ACM, Washington DC, USA (2004)
5. Whaley, J., Martin, M.C., Lam, M.S.: Automatic extraction of object-oriented component interfaces. SIGSOFT Softw. Eng. Notes **27** (2002) 218-228
6. Lewis, B., Ducasse, M.: Using events to debug Java programs backwards in time. Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. ACM, Anaheim, CA, USA (2003)
7. Lange, M.: It's Testing Time! Proceedings of the 6th European Conference on Pattern Languages of Programming and Computing, Irsee, Germany, UVK Universitätsverlag Konstanz GmbH (2001)
8. Institute for Information Systems, UCSD, University of California: UCSD PASCAL System II.0 User's Manual. (1979)
9. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical recipes in C: the art of scientific computing. Cambridge University Press (1988)

Social Networking Technologies for Free-Open Source E-Learning Systems

Francesco Di Cerbo, Gabriella Doderò, and Giancarlo Succi
CASE – Center for Applied Software Engineering
Free University of Bolzano/Bozen
Piazza Domenicani, 3
I-39100 Bolzano-Bozen,
{name.surname}@unibz.it
WWW home page: <http://www.unibz.it>

Abstract. In this paper, we illustrate our methodology for academic teaching, based on cooperative learning paradigm, which also relies on cutting edge e-learning techniques. We use Web 2.0 resources to fulfill requirements for an interactive-constructivistic “learning space”, where blended-teaching paradigm may be proficiently applied. We extend an existing Free/Open Source Software Learning Management System, to create a cooperative and community-based learning space adherent to our proposal.

1 Introduction

Much emphasis has been put in the importance of a constructivistic approach to achieve effective learning at all levels, including the academic one. The so-called “Dublin descriptors” adopted in many European universities as common educational indicators stress the importance of coupling the “knowledge” with the “applied knowledge” within reference environments, and across Europe, academic teaching methodologies are now promoting deep restructuring of curricula. In fact, the development of the so-called “soft-skills” has become an indispensable complement to any discipline, be it technology oriented or in the humanities. Pedagogues have already identified methodologies which may be applied in order to achieve such results; they are collectively called “active learning”. Examples are “problem-based learning” ([1]), “troubleshooting” ([2]) or “case-based teaching” ([3]).

Several pilot experiences already support this claim (see [4] and [5]), even if an exhaustive experimentation involving e-learning specialists and students at various levels and disciplines is not yet completed.

The development of a suitable technological infrastructure to support such activities has been mostly concentrated on the possibility to pass from the learning environment concept (the status-quo of the market) to that of “dynamical learning space” (DLS). In a DLS, people involved in learning are identified by their specificities, and meet each other to collaborate to the negotiation and construction of knowledge. The implementation of a DLS, giving importance and attention to the individuals inside a distributed process of learning, is our main goal.

Please use the following format when citing this chapter:

Di Cerbo, F., Doderò, G. and Succi, G., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 289–297.

The concept of “dynamical learning space” is not originating in the virtual environment by itself. Indeed, in a real (not virtual) learning space, both teacher and his/her class participate to the creation of a shared knowledge; they build up meanings and concepts where every individual has own importance inside the process. Teachers have the lead of the community, steering global effort towards learning targets, and every student may/must contribute.

Such a process requires very skillful teachers, able to involve learners’ community, letting everyone get his/her own role inside the teamwork. In case of a virtual teaching, such as in e-learning scenarios, a teacher should rely on a set of software tools, letting him involve his/her class, like during lessons in presence. Our software helps the teachers to coordinate such virtual communities, specifically at interaction level, in order to form a social network.

We developed our software in order to make possible this co-construction of learning materials and concepts by means of virtual experiences. We find that such approach can be useful not only in virtual (e-learning) classes, but when lessons are given both in presence and in virtual systems (blended learning). Such software also gives a precise and “automatic” track of activities conducted, without the distractive effect caused by recording them by hand. In this way, students and teachers may concentrate on learning process, without caring too much about activities tracking. A more detailed description of the pedagogical design is available in [6].

In Section **Error! Reference source not found.** we describe the choice of tools, and interfacing functionalities to augment them according to the “DLS” paradigm. In Section **Error! Reference source not found.** the underlying technologies are briefly described. Conclusive remarks are presented in Section **Error! Reference source not found.**

2 Tools

Inside an e-learning scenario, the concept of DLS naturally translates into that of a virtual environment, where interactions are welcomed and eased, and where every community service, like wikis and forums, contributes to the creation of a common knowledge as part of a structured learning process. Here, students and teachers are free to operate and move in a sort of virtual reality: a place, coherently with new Web 2.0[7] issues, where to put opinions or contents, to meet the class, even to find amusement, without a fixed interaction stereotype.

To this aim, previous research in Human-Computer Interaction (for instance, the work of Gräther and Prinz [8]) on community and presence awareness, and especially the concept of *social translucence* ([9]), were exploited. Functional interaction metaphors and schemes of services oriented to social networks were analyzed, where theories of cooperative learning and social constructivism can be put into practice.

But should such a new concept like the DLS be implemented from scratch, or can it be superimposed on top of some existing LMS? Existing infrastructures and

services were evaluated with respect to suitability to social constructivism. Free/Open Source Software solutions were especially considered because of the availability of particular license agreements, which allow users to legally and easily extend software, under certain conditions (see [10],[11]).

What is important to remark, here, is the freedom of use that such software license gives to licensee; a user is entitled to use, to study, to modify, and to redistribute a derivative work made upon original one. Our analysis included both a technical evaluation and a qualitative-strategic one, with the aim to understand the extensibility, the quality of the implementation, size, and support provided by the international community. As a result, we chose the Learning Management System Moodle [12] as our privileged software environment.

For Moodle, as well as for any other LMS, the introduction of a DLS implements a new social networking model. since every user is associated to an avatar, free to move in a web page, interacting with other users and with resources (for instance, opening course material or a real time chat). Logical proximity of activities is naturally mapped into physical proximity of the avatars in the virtual space. The possibility of using a spatial metaphor has proven succesful in the past (e.g. [13]).

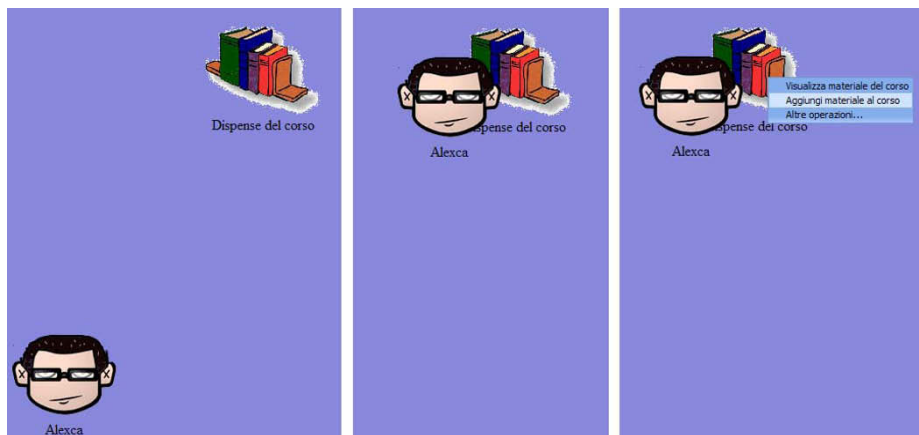


Fig. 1 Example of an interaction between an avatar and course material

A number of services of interest, like forums and wikis, are already found in Moodle. We added a videoconferencing system based only on Java (applets and services) in a client/server architecture, and an AJAX ([14]) whiteboard, shared among the community. Both are browser independent and do not require additional software installations at client side.

Figure 1 shows an example of an interaction between an avatar and a course material, for instance a book available to students. The avatar moves close to the

icon representing materials, and a pop-up menu appears, showing next possible actions. This happens also in case of interactions between avatars, in order to open a private chat session. The software development process we followed includes strong involvement of pedagogues in the choices taken by developers. Early feedbacks from them guarantees the highest quality and usability for the customers of the final product.

3 Implementation

Our software consists on two main subsystems: one implements all the operations at client side, while the other is responsible for managing all the interactions between all the users and the system. We defined a set of system behaviors, called *actions*, regarding possible interactions between clients and server. Then, we also defined a set of widgets and utility objects to be used at client-side. Each object is associated to a number of actions that are needed to perform its own tasks. These definitions have been formalized in XML documents, in order to be independent from any programming language (neither client- nor server-side). We refer to these elements as “XML Interfaces”. A representation of the two subsystems as well as XML interfaces is available in Figure 2.

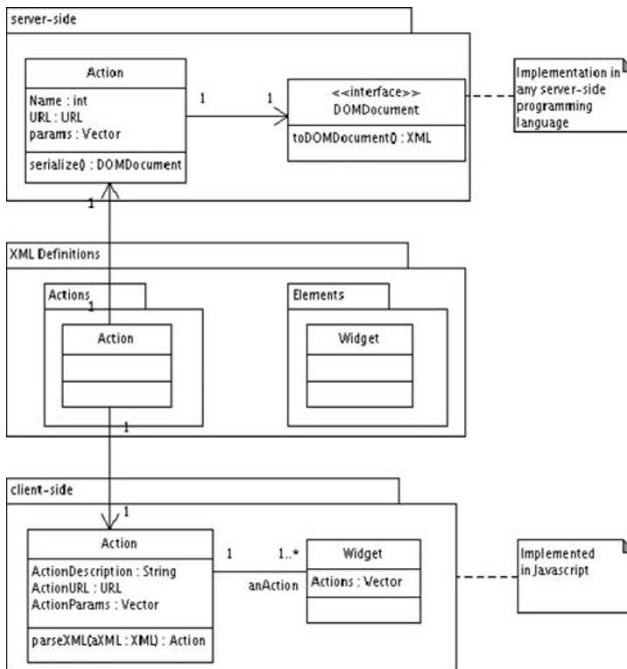


Fig. 2 Subsystems design

The development of objects for both sides was based on such XML interfaces; in fact, we designed and developed class definitions that are able to cope with specified exchange messages in XML format, if they can parse XML interfaces to produce accordingly XML requests. In this way, we are not bound to any programming language or software package.

Two special entities are responsible for communications between client and server: Status Manager and Status Monitor. Both of them embed the concept of “status”; as we aim at realizing a tool focused on live social interactions, we require that all the users share the same experience. Every user is cooperating with others, for instance by moving an avatar in response to another movement. As a consequence all users also have to share the very same status; we implemented this requirement keeping the state of the whole application server-side, and exchanging periodic updates with clients. Once an action has been performed by one of the clients, all the others are being notified within 1-2 seconds. This mechanism is based on the mentioned Status Monitor and Status Manager. Status Monitor is an object managing all the client side tasks and operations. In fact, it is responsible for creating the initial environment (avatars, rooms, widgets and so on), and to track any modification to its internal status (i.e., movement of the user's avatar, insertion of a

new message in the real-time chat and so on). Once a modification happens, Status Monitor sends an update message to its server-side counterpart, Status Manager.

There are as many instances of Status Monitor as there are clients, while the system has one and only Status Manager, as we need a central coordination point for the whole system. Then, Status Manager sends an update message at predefined intervals; this message is created considering the past history of all the Status Monitors since the last update message.

Figure 3 provides a sequence diagram showing interactions among subsystems.

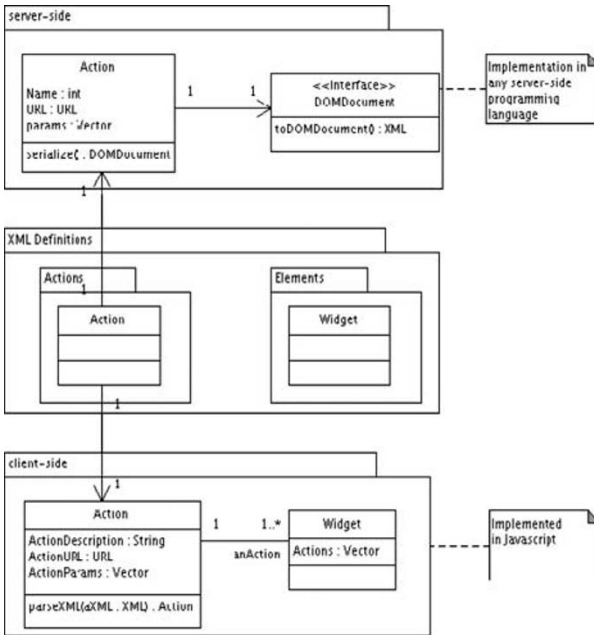


Fig. 3 Sequence Diagram for main objects: Status_Monitor and Status_Manager

XML Interfaces

In our system, interfaces are defined as XML Schema documents; they can be used to verify if a document is well-formed. We adopted XML Schema in order to enable verification of messages in our system, and to keep interfaces independent from actual language used in system implementation. Table 1 shows a sample action request.

Table 1: An example of XML Action message

```
<?xml ... ?>
<action name="aName">
  <param name="1"/ value=2>
  <param name="2"/ value=2>
  <url>

  </url>
</action>
```

Client-side implementation

Required objects and graphical widgets have been implemented in Javascript, the programming language we chose at client-side. Our main issue was delivering browser-independent code. In fact, different browsers often yield different behaviors for Javascript code, both in the graphical part, and in the invocations to asynchronous methods, that are the base of AJAX paradigm. For these reasons, we adopted a Javascript framework to encapsulate and provide us portable solutions for all the issues mentioned: the Dojo toolkit

Dojo (available from www.dojotoolkit.net) is an Open Source DHTML toolkit written in JavaScript. It supports asynchronous calls (i.e., all AJAX interactions), as well as unit testing and layering, with a very small core which provides the package system. The Dojo toolkit enables our software to be browser-independent, as it detects the type of browser where it is executed, and automatically adapts the actions it is required to perform to the actual environment.

Server-side implementation

We implemented our server-side subsystem using PHP, due to the choice of Moodle as our base LMS. PHP version 5 allowed us to exploit a number of services, including reflection and full support for Object-Oriented programming, that we needed. Moreover, as Moodle uses PHP, this simplifies the deployment process, without requiring to set up an heterogeneous system to use our software.

4 Conclusions

We have described the pedagogical approach, tools chosen, and implementation technologies that we have been employing in order to support social networking inside a new generation LMS. The system extends the well-known Moodle Free/Open Source software with a “dynamical learning space” concept, providing a novel interaction model, especially suited to support blended learning. At present, the system has been deployed on Moodle 1.8.3, and we are starting to extensively experience it with users (university teachers and students).

To evaluate our solution, we have started to use automatic tools for collecting process metrics ([15]) already available and developed at the Free University of Bolzano Bozen ([16],[17]). This will allow us to discover usability patterns inside users' experience in an objective way. It is possible, for instance, to understand what services are the most used, for how much time, and the “productivity”, in terms of outcomes deriving from user's experience with that specific tool. This would provide a base of evidence useful on two different scenarios: a technical one, to analyze introduced improvements and eventually refine them, and the pedagogical one, to bring quantitative confirmations for qualitative considerations.

Using both feedbacks coming from educational specialists and quantitative measures, it shall be possible to formulate a complete report of our experience, that in conjunction with a new Learning Management System platform would result in a complete educational proposal, a framework able to coordinate and to combine both in-presence and virtual lessons.

Bibliography

- [1] Ward, J.D. and Lee, C.L., A Review of Problem-based Learning, *Journal of Family and Consumer Sciences Education*, Vol. 20 , n. 1, 2002,
- [2] Jonassen, D.H., Hung, W., Learning to Troubleshoot: A New Theory-Based Design Architecture, *Educational Psychology Review*, Vol. 18, n. 1 , 2006, DOI: 10.1007/s10648-006-9001-8
- [3] Jonassen, D.H., Hernandez-Serrano, J., Case-based reasoning and instructional design: Using stories to support problem solving, *Educational Technology Research and Development*, Vol. 50 , n. 2 , 2002, DOI: 10.1007/BF02504994
- [4] Stefanova E., Sendova E., Nikolova I., Nikolova N., When I*Teach means I*Learn: developing and implementing an innovative methodology for building ICT-enhanced skills., 2007, *Proceedings of the IFIP Conference Informatics, Mathematics, and ICT: a ‘golden triangle’ IMICT 2007*, pg., Northeastern University, Boston, MA,
- [5] Dodero G., Ratcheva D., Stefanova E., Miranowicz M., Vertan C., Musankoviene V., The virtual training center: a support tool for teachers community, 2007, *Proceedings of Balkan Conference in Informatics (BCI 2007)*, pg. , Demetra Ltd,

- [6] Di Cerbo, F., Succi, G., A proposal for interactive-constructivistic teaching methods supported by Web 2.0 technologies and environments, 2007, Proc. of 18th International Conference on Database and Expert Systems Applications, 2007 (DEXA '07), pg. 648-652, ,
- [7] O'Reilly, T., What Is Web 2.0, 2005, <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>
- [8] Gräther, W. and Prinz, W., Supporting the Social Side of Large Scale Software Development, 2006, Supporting the Social Side of Large Scale Software Development, pg. , Microsoft Research,
- [9] Erickson, T. and Kellogg, W.A., Social translucence: An approach to designing systems that mesh with social processes., Trans. Computer-Human Interaction, Vol., n. 7, 1 , 2002,
- [10] Stallman, R.M., Free Software, Free Society: Selected Essays of Richard M. Stallman, 2002
- [11] Raymond, E., Cathedral and Bazaar, 1999
- [12] Dougiamas, M., Taylor, P.C., Moodle: Using Learning Communities to Create an Open Source Course Management System, 2003, roceedings of the EDMEDIA 2003 Conference.
- [13] Pfister, H.-R., Wessner, M., Beck-Wilson, J., Miao, Y., & Steinmetz, R. (1998). Rooms, protocols, and nets: Metaphors for computer supported cooperative learning of distributed groups. In: Proceedings of the ICLS'98 - International Conference of the Learning Sciences (pp. 242-248). Charlottesville, VA: AACE Association for the Advancement of Computing in Education.
- [14] Garrett, J., Ajax: A New Approach to Web Applications, , www.adaptivepath.com/publications/essays
- [15] Humphrey, W., Introduction to the Personal Software Process, 1997
- [16] Sillitti A., Janes A., Succi G., Vernazza T., Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data, 2003, Proceedings of EUROMICRO 2003, pg. 336-342,
- [17] Scotto M., Vernazza T., Sillitti A., Succi G., Managing Web-Based Information, 2004, Proceedings of ICEIS Conference, pg. 575-578.

The Networked Forge: New Environments for Libre Software Development¹

Jesus M. Gonzalez-Barahona¹, Andrés Martínez², Alvaro Polo², Juan José Hierro², Marcos Reyes², Javier Soriano³, and Rafael Fernández³

¹ GSyC/LibreSoft, Universidad Rey Juan Carlos

jgb@gsync.escet.urjc.es

² Telefónica Investigación y Desarrollo

³ Universidad Politécnica de Madrid

Abstract. Libre (free, open source) software forges (sites hosting the development infrastructure for a collection of projects) have been stable in architecture, services and concept since they become popular during the late 1990s. During this time several problems that cannot be solved without dramatic design changes have become evident. To overcome them, we propose a new concept, the “networked forge”, focused on addressing the core characteristics of libre software development and the needs of developers. The key of this proposal is to re-engineer forges as a net of distributed components which can be composed and configured according to the needs of users, using a combination of web 2.0, semantic web and mashup technologies. This approach is flexible enough to accommodate different development processes, while at the same time interoperates with current facilities.

1 Introduction

The libre (free, open source) software² development community, considered as a whole, is one of the largest cases of informal, globally distributed, virtual organization oriented to the production of goods. Hundreds of thousands of developers (working for companies or as volunteers) share a large base of source code (hundreds of millions of lines of code) and knowledge which they use to produce and improve software products. This collaboration has been possible only thanks to the intensive use of Internet-based tools, currently offered mainly by development forges such as SourceForge.

¹ This work has been funded in part by the European Commission, through projects FLOSSMetrics, FP6-IST-5-033982, and Qualipso, FP6-IST-034763, and by the Spanish Ministry of Industry, through projects Morfeo, FIT-350400-2006-20, and Vulcano, FIT-340503-2006-3

² In this paper the term “libre software” is used to refer both to “free software” (as defined by the Free Software Foundation) and “open source software” (as defined by the Open Source Initiative).

Please use the following format when citing this chapter:

Gonzalez-Barahona, J.M., Martínez, A., Polo, A., Hierro, J.J., Reyes, M., Soriano, J. and Fernández, R., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 299–306.

In fact, the dawn of libre software development communities is linked to the spread of the Internet. Since their startup, many development teams used Internet-based tools for collaboration, and dissemination of the produced software. With the spread of the web they were integrated in ‘project sites’ which provided the infrastructure used for collaboration [7]. Around 1995 those sites provided mailing lists, download areas, issue tracking, source code management (usually CVS), and static HTML pages with information and documentation about the project.

During late 1990s and early 2000s some organizations started to offer those facilities to large collections of projects. The most known, and by far the largest of them, is SourceForge, established in 1999, but many others do exist. Many of them run different forks of the original SourceForge software, of which the most popular (specially for small sites) is GForge. In addition, large projects (GNOME, KDE, Apache, Eclipse, OpenOffice.org, Mozilla, etc.) maintain their own forges, usually with ad-hoc software, and similar systems are used as well in corporate environments for the development of non-libre software [3].

The basic architecture and services of all these forges have evolved only slightly during the last decade, meeting many of the needs of development teams, and proving to scale well to the tens of thousands of projects, hundreds of thousands of developers, and hundred of millions of lines of code. However, they also show several problems, rooted in their incomplete adaption to the extremely distributed, interrelated and flexible nature of the libre software community. To fix them, we decided to rethink the concept of forge.

2 Current forges and their problems

The architecture of current forges is similar [8, 1], with the needed considerations for scale, which may lead to the use of computer farms in large cases. Their main components are a web server (as front-end), a web application (e.g., GForge, providing the specific services of the forge), an SQL database (persistent storage for non-massive elements) and some specific components (such as source control or mailing lists management).

The web application provides services such as: information about projects and developers (including authentication), issue tracking, news and forums, wikis, scheduling services, and a common web interface to most of the functionality (including downloads of code). In addition, specific components usually found accompanying this application are: a download manager, a source control management system (CVS, Subversion, etc.), and a mailing lists management system with archiving facilities.

About one decade of intense use of these forges have uncovered several problems:

- They are project-centric instead of developer- or user-centric. The service “unit” is the project, but both libre software developers and users are usually interested in many projects, hosted in several forges.
- Monolithic approach. Each forge offers a fixed set of services, each implemented by a specific system, which impedes that each project chose the software they prefer for every service, and slows down innovation (only administrators can add new modules).
- Isolation. For most practical purposes, each forge is isolated from others (even with federation facilities).
- Poor integration. Each separate component offer its own user interface.
- Lack of fine-grained coordination. Related elements in different subsystems are difficult to relate to each other (e.g., patch to fix a bug and its revision in source code management repository).
- Lack of support for views. A given project, or collection of projects, cannot offer multiple views to users.
- Little attention to collaborative knowledge sharing. Collaborative tagging, bookmarking and cataloguing, for instance, are missing.

There are also some concerns related to the extreme concentration in the most popular forges. Some of them are becoming single points of failure and potential control for libre software development as a whole [4, 5]. Therefore, they become a critical infrastructure that has to be maintained and defended something which is intensive in human resources, and difficult to scale.

A number of solutions to some of these problems have been proposed, either as improvements to existing forges, or as new systems (such as Launchpad), but they still remain open issues needing a comprehensive approach. In some reviews of future developments in the field [9], several scenarios that would address some of these problems are also described, but they do not propose detailed designs or implementations.

3 The networked forge

Given this situation, we found it reasonable to re-engineer the fundamentals of forges, under the following main lines: aggressive distribution (services located in different sites); easy relocation (backup and restores interfaces that allow for quick recovering of a service at another location); user-centric scenarios (which allow developers and users to access easily all the services they need, despite the project to which they are related); fine-grained links between different services (so that developers can access related information easily); federation and presence in different forges (so that a project can be supported by several sites); relationship with upstream and related projects (even when residing in different forges); composability of independent elements; and fostering the development and sharing of innovative

component. As an important side-target, we also wanted to maintain a high level of interoperability with legacy services, to ease the transition.

Our proposal, the networked forge, is based on the idea of considering forges not as single sites providing a monolithic set of services to host projects as isolated silos of knowledge, but as a collection of distributed components providing services, among which knowledge is shared. Each project decides on its own customized set of services, and users can configure their own working environment. This idea matches that of mashups [6] or semantic web 2.0 applications [2].

The main components of the architecture are (see figure 1 **Error! Reference source not found.**):

- Integrated services, specifically built to feed into the proposed framework.
- Legacy systems, that have to be integrated in the networked forge.
- Client components provide the user interfaces.
- Connectors and adapters, connecting legacy components to the rest of the system (connectors are used with those components providing a semantic, RDF-like interface, adapters with non-semantic components).
- Aggregators, collecting RDF channels and processing them in several ways.
- Locators, used as name services, allowing for the registration of specific components.
- Catalogues, in which components available for a certain community are registered.

Several connectors or adapters can work with the same legacy service, providing different interfaces to it. Conversely, a given connector or adapter can work with several instances of the same kind of legacy service, providing the same interface to several sites. Client components can interact directly with the integrated services, with some semantic legacy services, and with connectors, adapters and aggregators. Aggregators can interact as well with all these components.

The architecture imposes little limitations to where the different components may reside. Usually, legacy services will be hosted in different sites in the Internet. Integrated services will also run somewhere in the Internet, but they could even be located in some cases in the user workstation. Client components will usually be hosted in some web server in the Internet, but will run in the user browser. However, other combinations are possible: they could also run in servers and be visualized in web browsers, or reside in the client side, for instance as modules for an IDE.

The communication between all the non-legacy components is performed with HTTP and WebDAV, with all the components providing RDF channels via REST interfaces. The architecture uses semantic web technologies for the exchange, handling and querying of the data between the different components of a forge network.

All connecting components feature a common REST interface, which simplifies composition. The information provided by the component is always an RDF channel, including semantic labeling, which is obtained via HTTP. Aggregators act as filters, also accepting RDF channels as information.

Components provide RDF channels, but no other interface intended for end users (such as HTML pages). Therefore, the user interface is usually provided in the client side. When the client application is a web browser, AJAX techniques are used to display the data in useful ways. In fact, the most natural way of producing an application interface for a networked forge will be via a mashup (including those implemented using Google Gadgets or Yahoo Pipes). But it is important to notice that any kind of application could interface to the system from the desktop, including for instance RDF client-side aggregators, or components for an IDE such as Eclipse plugins.

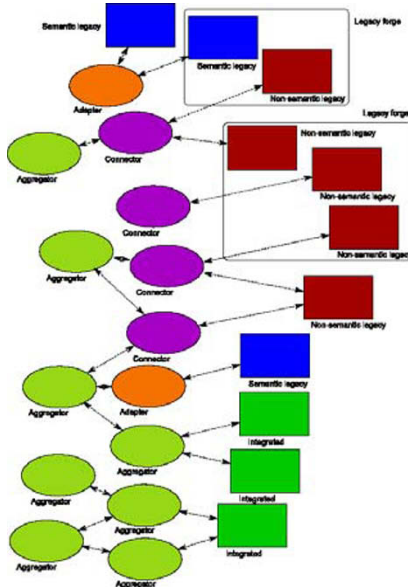


Fig. 1: General architecture. Clients, locators and catalogues are not shown to improve visibility.

Locators and catalogues are optional but important components. Both accept, at configuration or run time, information about the location of the different components (that is, their url), and provide upon request that information (either for all the components registered, or for a certain subset of them fulfilling some property).

Catalogues maintain information about available components for forges, while locators are configured to ‘build’ a networked forge: the components of the forge will be those registered with the locator. Therefore, components will be a part of several networked forges just by being registered with different locators.

To be functional, the services offered by the components of the networked forge are integrated in flexible and diverse ways. This integration can be accomplished either in the server side (using aggregators) or in the client side (using mashups, standalone RDF aggregators, or plugins in an IDE).

4 Implementing the concept

We have implemented a first version of the networked forge. A proof-of-concept setup using it is composed of several connectors and adapters to integrate legacy components (GForge, Trac, Subversion, etc.), several widgets (using Google Gadgets technology) to implement a client-side application to access the networked forge, and several locators. Figure 2 **Error! Reference source not found.** shows a screenshot of this implementation.

Each widget reacts to changes in other widgets. For instance, the “My Vulcano” gadget (which shows a list of projects) controls the content of the “Project Details”, “Project Tickets”, “Project’s Wiki” and “SVN Log” widgets. Widgets can also be configured by the user, pointing them to different urls: “My Vulcano” widget shows a different list of projects if pointed to a different locator.

The current implementation is written in PHP, Python and Java (standalone adapters or connectors on server side), and in JavaScript (client side adapters and connectors and gadgets logic). It is still minimalistic, but even so it shows the great potential of the concept of a networked forge.

Other proof-of-concept scenarios mimic the functionality of a legacy multi-project forge; the forge for a company, including its collaborations in libre software projects; the forge for a software distribution, including the development of upstream (original) products and their packaging activities; and a personal forge.

5 Conclusions and further work

Forges are an increasingly important tool for collaborative software development. However, the traditional model for implementing them has some shortcomings that we address with a new design: the networked forge. Networked forges allow for the easy integration of components residing in many different sites and administrative realms, while at the same time provide a great deal of flexibility.

Given a networked forge infrastructure, users, developers, projects, companies and other actors can configure the forges they need, share their configurations, and select the exact services they prefer. Interoperability with legacy systems is a part of the model, as well as a seamless integration with many different client-side environments, from web-based mashups to traditional IDE applications. The proposed architecture has been tested in proof-of-concept implementations.

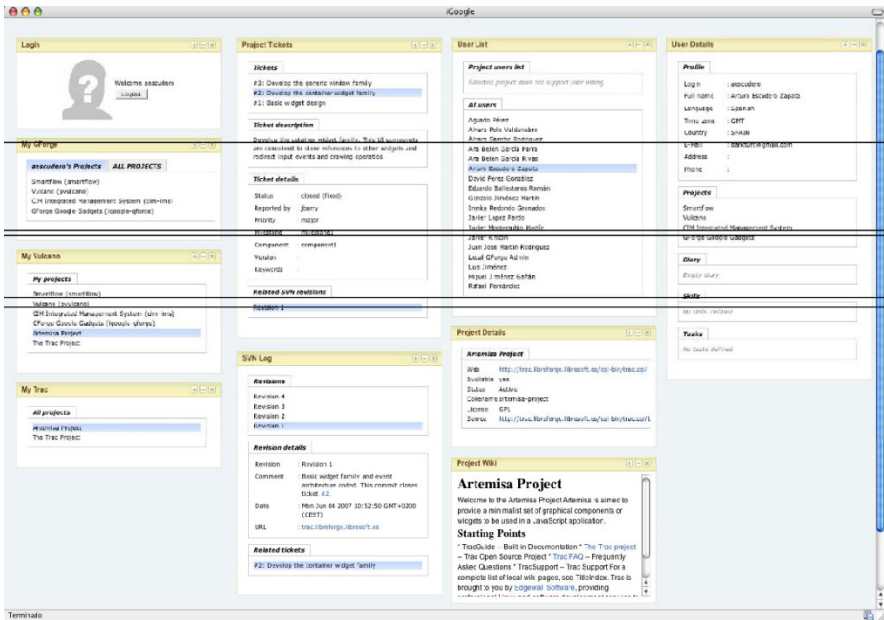


Fig. 2: Proof-of-concept networked forge. User interface implemented with Google Gadgets.

Some aspects of the design still remain open, while we explore several options for them. Security and authentication, for which we are studying single sign-on or common authentication infrastructure technologies, are two of them. Selection of ontologies for the exchange of information, and formats for specifying connections between components are also an open field. We are also developing more complete implementations in the context of the Vulcano and Qualipso projects.

In summary, the networked forge is a concept that suits well the practices of the libre software development community, and can also be applied in other more traditional environments.

Acknowledgements This work has benefited from many discussions in the context of the Vulcano, Morfeo, OSOR, FLOSSMetrics and Qualipso projects. We thank all the persons who contributed with ideas, comments and criticisms.

References

[1] Olivier Abdoun, Bernard Lange, Stéphane Laurière, Irenka Redondo, and Christian Rémy. Collaborative development environment: A state of the art. Technical report, Qualipso Project, 2007 (to be published).

- [2] Anupriya Ankolekar, Markus Krötzsch, Thanh Tran, and Denny Vrandečić. The two cultures: mashing up web 2.0 and the semantic web. In *Proceedings of the 16th international conference on World Wide Web*, pages 825–834, New York, NY, USA, 2007. ACM Press.
- [3] Grady Booch. Introducing collaborative development environments. Technical report, IBM, December 2006.
<http://www.alphaworks.ibm.com/contentnr/cdepaper>.
- [4] Loïc Dachary. SourceForge drifting, 2001.
<http://fsfeurope.org/news/article2001-10-20-01.en.html>.
- [5] Jesus M. Gonzalez-Barahona and Pedro de las Heras Quirós. *Future Trends in Distributed Computing*. Andre Schiper, Alex Shvartsman, Hakim Weatherspoon, Ben Zhao eds., chapter Hosting of Libre Software Projects: A Distributed Peer-to-Peer Approach, pages 207–211. Number LNCS 2584 in Lecture Notes in Computer Science. Springer Verlag, 2003.
- [6] Anant Jhingran. Enterprise information mashups: integrating information, simply. In *VLDB'2006: Proceedings of the 32nd international Conference on Very Large Data Bases*, pages 3–4. VLDB Endowment, 2006.
- [7] Tim O'Reilly. Lessons from open-source software development. *Commun. ACM*, 42(4):32–37, 1999.
- [8] Patrice-Emmanuel Schmitz, Abdelkrim Boujraf, Rishab Aiyer Ghosh, Emidio Stani, Juan José Amor Iglesias, Julia Anaya González, and Álvaro del Castillo San Félix. Feasibility study. Technical report, Open Source Observatory and Repository, IDABC, 2007 (to be published).
- [9] Jim Whitehead. Collaboration in software engineering: A roadmap. In *Future of Software Engineering (FOSE '07)*, pages 214–225, 2007.

To What Extent Does It Pay to Approach Open Source Software for a Big Telco Player?

Massimo Banzi¹, Guido Bruno², and Giovanni Caire³

¹ Telecom Italia - Technology & Operations,
Innovation Architecture and Quality dep.,
Via Zambra 1, 38100 Trento Italy
Free Univerisity of Bolzano-Boozen Italy
massimo.banzi@telecomitalia.it

² Telecom Italia - Technology & Operations,
Innovation Architecture and Quality dep.,
Via Reiss Romoli 274, 10148 Torino Italy
guido.bruno@telecomitalia.it

³ Telecom Italia - Technology & Operations,
Innovation Architecture and Quality dep.,
Via Reiss Romoli 274, 10148 Torino Italy
giovanni.caire@telecomitalia.it

Abstract. In this paper we describe the strategy under adoption in Telecom Italia (TI) Technology Department toward open source software. This stems from trying to create synergy among big Telco Player to increase knowledge and influence over strategic communities to the evaluation of the creation of new communities over internally developed applications. In particular here the approach and the expectations in starting the community on WADE (Workflow and Agent Development Environment) is described. This is a platform used to develop mission critical applications and is the main evolution of JADE a popular Open Source framework for the development of interoperable intelligent multi-agent systems. It adds to JADE the support for the execution of tasks defined according to the workflow metaphor as well as a number of mechanisms that help managing the complexity of the distribution both in terms of administration and fault tolerance. The idea is to use WADE as a mean to gather critical information on the opportunity of approaching OS as a strategic mean toward the development of always more important application in Operating Support System for TI, possibly also involving other great Telco Players For this reason great care is being paid in setting up the Community environment and in deciding which metrics are to be extracted from it, since the result will be the input for a strategic decision in TI.

Please use the following format when citing this chapter:

Banzi, M., Bruno, G. and Caire, G., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 307–315.

1 Introduction

What is the value in OS for a Big Telco Player such as TelecomItalia?

With 7.3 million broadband connections (retail and wholesale) 0, Telecom Italia is currently the leading operator in the national TLC market. It has one of the most penetrating and advanced network in Europe, with an extension of over 107 million Km in copper lines (access) and 3.8 million km of optical fibers (transport and access).

This is a question that is intriguing OSS Software Factory Management in TI Technology Department. Nowadays the constant search for reduction of Total Cost of Ownership (TCO) is forcing more and more to find solutions effective, but suitable to fulfill all OSS processes in a modern big Telco Company (Service Delivery, Network Creation and Inventory, Workforce Management, etc.). The analysis toward Open Source Software has been obviously anticipated by initiatives such as the Service Rationalization and the Application Rationalization, to reduce redundancy of services and tools, but the obvious consequence has been the thought whether we could substitute Commercial tools with OS ones without losing confidence and efficiency.

The first approach toward OS has been driven by a set of indicators that have been formalized as follows 0:

- ▶ The product has to fulfil a need and provide a well defined and understood service so that it can become a commodity.
- ▶ It has to be a leader on the market in its context
- ▶ There should be a high number of “customers” that are also “contributors” for the product
- ▶ It has to be modular
- ▶ It must have a robust and well designed core
- ▶ The community needs to be large and talented.

This was the theory.

What actually happened was that, also according to several surveys 0, the core of *use of OS in both mission-critical and non-mission-critical IT environments in TI rate higher for infrastructure than application software*, and infrastructure software uses OS components in conjunction with proprietary software.

More, in TI the use of OS applications in OSS vertical solutions is almost absent.

The *reason stands mostly in the opportunistic adoption of OS* in every single case, not driven by a real strategy.

In other words, in using OS software, an “opportunistic” approach (simply just use it) can be adopted or “a strategic one”.

In the first case OS can be selected eventually also just as a proof of concept (using it simply as a leverage against COTS vendors). Or it can be simply used, directly, without even leaning on the community: it is downloaded, embedded in proprietary software and fully managed as an integral part of the solution.

Clearly in this case the community is used simply as a source of information for solving possible problems that can arise. No contribution is given.

In the second case two situations can arise

- if a community and a suitable OS solution considered as important does exist, it is necessary
 - ▶ the exploitation of community as a way to reduce development costs, accelerate the availability of new features, keep the product more robust through the high number of users and contributors
 - ▶ since the product is probably to become an integral part in the player products portfolio, strong Competence Centres participating the communities are a necessity to be able to influence selected strategic communities and the roadmap of the products.
 - ▶ to keep a strong link between the innovation of the product in the community and the Telco production itself (basically an internal development branch has to be kept to validate every new added feature in the context the OS product is used by the company).

Clearly all these can be achieved only by actively participating the community, injecting it with resources and competences..

- If a community and a suitable OS solution does not exist, it is possible to evaluate the possibility of “open” software solution internally developed or to create a new community from scratch, joining several stakeholders needs.

Up to now TI just used the opportunistic approach. But now things are changing. The main reason are the arising consciousness that OS ensure best-of-breed solutions, improved ROI, and vendor independence and flexibility.

Moreover OS solutions reached a quality comparable with that of proprietary Software, and the possibility to lean upon third companies providing support on OS solutions thus reducing risk in OS adoption are removing the remaining doubts.

2 Different approaches

According to the above consideration, TI Technology department decided to approach OS with a real strategy, without also rejecting the possibility of an opportunistic approach for non strategic solutions.

2.1 Synergy among big Telco players on existing communities

As said, in case the community exist, the Player typically uses the OS product directly or mediated by a broker upon which it unloads part of the risks.

Sometimes it also acquires from vendor or even brokers customized OS software. For whatever choice of OS adoption strategy, needs in OSS for most Telco operators are similar, and the competition among Big Players is on services not on software solutions.

So a question arose: why not to coordinate the interaction with OS communities?

This can result in several benefits, first of all the influence on the community can be stronger but also if brokers are used, these can be shared; contributions can be distributed as well as competences and the weight on board of the community can be stronger.

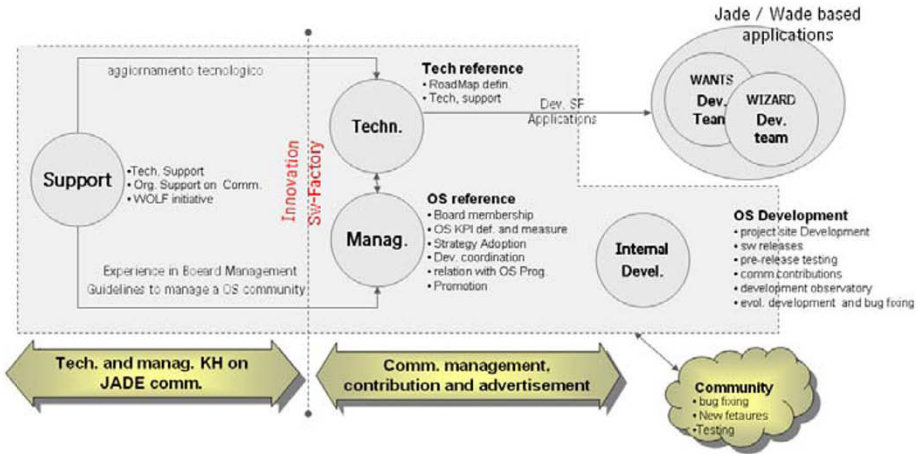
Finally a common action on the communities, on brokers, but also on vendors can result in the possibility of defining standards instead of undergoing them.

Hence the idea of proposing a working-table among Telco Players to promote and manage the creation of community specific Working Groups that homogeneously interact with the community, eventually enforcing and promoting single player initiatives on OSS.

2.2 Creation of a new community

If a suitable community does not exist, maybe it can result profitable to share an internally developed solution to extend the test base or also the development one if enough interest can be excited on the solution. This is the case of WADE where a solution based upon an existing Open Source platform (JADE 0) used to develop internal critical application has been chosen as a laboratory to verify the opportunity of such an approach.

The model that we decided to adopt is depicted in 0 .



Organizational scenario

There, a Support Group has been identified, which is the original development team of WADE that clearly owns the major skill on the solution. It is engaged also in other activities and so it is expected to provide only a high level support to the initiative. A Technical manager for the specific solution has been identified within the Factory (the department in TI that actually develops or integrates software solutions). He currently is the responsible for all the applications developed on the top of WADE. He will also be responsible for deciding on the roadmap of the product within the arising community.

There is also a management role which is not just for the community created on WADE, but that is expected to manage all the solutions that will be opened. He will be responsible for the shared environment, the collection of metrics on it, its promotion, the coordination of the resources allocated on the community that will have to deal with internal builds, but also with releases, with the validation of community contributions, bug fixing, main evolution, etc.

The expectation is that we will be able to engage other companies in the initiative so that, others join the structure described in 0. Another technical reference is expected to be identified in the “other” company to create a “WADE working group” as well as “other” developers to contribute and a Manager to be the only reference in the “other” company for the sharing of OS solutions between the two. The two managers will be also members of the Working-table described above.

2.3 Impacts

What are the impacts on overall structure of the development and what the actions to be taken into consideration?

It is clear that in the creation of a working table a lot is to be done on the mentality.

Apparently it is still rooted in many companies the idea that sharing collaboration with historical competitors can be a mistake. Give access to information and also the source code used to develop mission critical solution is a bad choice.

But the truth is that currently the standardization of OSS solutions causes few great vendors and consultants to share the market and provide different Telco Companies with the same solutions, eventually slightly customized.

What competitive advantage can be found in this? Better is consciously do the same thing but driving the choice, not being led individually to almost the same solution with multiplied costs.

What we are currently doing is the exploitation of already existing channels (TMF Operator Committee) to promote the initiative among big Telco Players and verify the interest on it.

In crating the new WADE community instead we have to face other problems. As said we want to be able to measure the impact that introducing OS can have either in the organization or in the quality of the software.

As clearly stated by 0, opening up a project can add whole new sets of complexities, and cost *more* in the short term than simply keeping it in-house. Opening up means arranging the code to be comprehensible to complete strangers, so **documentation** needs to be clear where sometimes it is almost absent.

Moreover **packaging** is a topic that is underestimated when the development is internal, but in distributing it, it is necessary to create easy installable packages.

More if interest arise on the community, there is the added burden of answering developers' questions for a while before seeing any benefit from their presence.

This activity is much closer to promotion than to software development, but sometimes it is the first impact on the environment, on the care devoted to users, on the possibility of having support to ensure a success to the project, more than the embedded idea that can be successful.

Also **management aspects** are very different from those necessary in in-house development.

There are several developers, notoriously independent, who do not accept easily the idea that behind the product stands a big company. Even if the community will be done of just resources provided by different companies (and this is undoubtedly limiting) there is a coordinating activity far more complex than the one required for a homogenous group of co-located developers.

There should be a management conducting an activity closer to that of a psychologist that have to keep different people together giving them the consciousness that working together is more productive than working individually.

An Open Source community has to be treated more as a social network than as a developers team. This is also the reason of the importance of the development envi-

ronment that must provide the easiest tools for communicating, sharing experiences and problems. In other words it has to be a place developers want to keep coming back to.

Finally *legal aspect* have to be carefully treated since one of the most diffused license for instance, the GNU Public License, as discussed in 0 , appears to have the ability to expose proprietary software developers to substantial risk if used improperly.

2.4 Metrics

The purpose of the initiative is that of “measuring” the opportunity of using OS as a resource for TI. This means that all possible measures, translated into “savings” has to be collected.

What saving means is clear, not as much how indirect saving can be collected and then monetized.

And in using OS there are not just savings, as repeatedly stated above, there are extra costs: the cost of setting up an environment to be accessed from outside, the cost of the restructuring the product to be open effectively. The cost of a parallel internal environment to reproduce the build before delivering the solution to internal projects, the cost of internal development and management, etc.

Clearly an expense has been activated to charge the effort to be devoted to such activities that are clearly identified, and this can rate the direct cost of the initiative.

Similarly we are collecting the effort spent in past years for maintaining WADE, for a comparison of the values.

But obviously we want also to measure the contribution given by the community, that cannot be measured in man/hours.

So the usual environmental metrics are collected 0:

- The number and growth of “core” members divided by dominion
- The number and growth of downloads
- The number and growth of contributions either to the source code repository or to the mailing list (divided in developers and users, this measures the “vitality” of the community)
- The mean time to respond to an issue (again full and cleaned by our direct contributions)
- The fixing rate (internal, but more important form external contributors)
- The main fixing time from external contributors (“Reaction Rate”)

As well as quality metrics:

- Delta times to manage the introduction of new features with respect to insourcing criteria
- Debugging times and quality
- Cohesion - change in cohesion - structural 0

- Bugs (normalized on LOC, bug fixing mean time) mainly on critical components (the more changed ones)

Also organizational metrics are to be collected:

- Measure the level of control and influence on the community: how many internally proposed new features have been accepted by the community and implemented by others than internal contributors

Clearly all these need to be translated into a real value for Telecom Italia and so apart from translating the indicators into objective numbers, it is necessary to find conversion parameters to turn these into saved effort.

We are still working on all this.

3 Conclusions

As widely described TI is trying the approach to OS as a possible way for further reduce the TCO of its OSS.

The strategic relevance of WADE and its choose as trial for such an initiative demonstrates the interest of TI to approach Open Source as a real resource for its business.

The process has just started: a plan for the initiative is ready and starting from March 2008 WADE will have its own environment, mailing list, forum , etc. and a web site from which it will be downloadable.

A document describing the full set of metrics that will be collected is under development in collaboration with the University of Milano and these metrics will be systematically collected to measure the effectiveness of the initiative and to refine the environment according to result as well as on feedback from the community.

Every three months these metrics will be published to the management who has great interest in the initiative.

Concurrently ha thread with TMF has been started with the goal of starting a catalyst project on OS to give the initiative all the strength of an organization that has also tha aim of proposing standards within Telco community.

It's a bet we expect to win.

References

The individual shareholder Guide Oct. 2007 http://ticlub.telecomitalia.com/pdf/Guida_1H_2007_ottobre_EN.pdf

Solari A. , Le opportunità del modello Open Source; un percorso in Telecom Italia - Technical Information Services verso il SW Open Source, OSBAIT Nov. 2006

Laurie Wurster, User Survey Analysis: Open-Source Software Summits, North America and Europe, 2006, Gartner

Rishab Aiyer Ghosh, Study on the Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies(ICT) sector in the EU, MERIT 2006,

Karl Fogel, Producing Open Source Software How to Run a Successful Free Software Project, <http://producingoss.com/>

JADE - Java Agent Development framework. <http://jade.tilab.com>.

Frost J., Some Economic & Legal Aspects of Open Source Software, 2005, <http://students.washington.edu/jjfrost/>

Ardagna C., Damiani E., Frati F. FOCSE an OWA-based Evaluation Framework for OS adoption in Critical Environment, in Open Source Development, Adoption and Innovation, Springer 2007

Misic V., Measuring the Coherence of Software Product Line Architectures, Tech. report TR 06/03 – Univ. of Manitoba

A Framework to Abstract The Design Practices of e-Learning System Projects

Alain Corbiere
LIUM-IUT, Le Mans University,
52, Route des Docteurs Calmette et Guérin,
53020 Laval Cedex 9, France

Abstract. The use of ALT (Advanced Learning Technologies) creates dynamic sharing and exchanging between open source communities that diffuse e-learning systems. In our opinion, the designer practices define new perspectives on e-learning design which are not structured and highlighted enough. This article shows the capabilities of a generic framework to analyse the design practices on a open source project and to explicit these practices. We describe how the semantics for architectural specifications proposed by RM-ODP (Reference Model-Open Distributed Process) framework were applied on an e-learning system project to analyse the principles of invariants, structural and functional.

1 Introduction

The current practices using ALT define new perspectives on e-learning design. First, the open source project applies a component-based software engineering [1] which makes it easier to specify and to reuse the software artefacts in design processes. Then, the e-learning designers are required to choose, understand and use the informational models provided by the IMS Global Learning Consortium to specify their decisions. The design intentions of an e-learning system are increasingly explicit.

These facts bring us to search for an abstract framework for the engineering and the re-engineering of software system and to apply on an e-learning system. This framework must define a set of terms to explicit the observations on the practice of the open source communities that design e-learning platforms. This framework must be sufficiently precise to analyse and to describe the code and the architecture, and sufficiently open to allow its own interpretation.

In this article, we propose the RM-ODP (Reference Model-Open Distributed Process) framework which was adopted as an ISO (International Organization for Standardization) standard. This framework combines the logic of a formal description to specify the architecture and to consider the distributed information technology evolution [2]. Moreover, the objective of the design of e-learning systems is described as “a collaboration between the disciplines of educational science and computer science” [12].

Please use the following format when citing this chapter:

Corbiere, A., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succì; (Boston: Springer), pp. 317–323.

This article illustrates the use of the framework by describing one interpretation of a four step specification on an open source project. We abstract the design practices of the e-learning system project and explicit the new perspectives based on the information model diffused by IMS Global Learning Consortium.

2 RM-ODP framework

This framework is defined as a meta-standard on the distributed system [3]. It is domain independent and supports a specification process. This tool allows us to describe the different design practices observed while respecting the vocabulary and to structure it according to its architecture. This framework provides two abstraction levels and it represents describes the transformations corresponding to the acts of specification, structuring and modelling.

The first abstraction level is defined by five viewpoints “each with an associated viewpoint language which expresses concepts and rules relevant to a particular area of concern in terms of which of a system can be described from that viewpoint” ([3] item 6.2.1).

- The enterprise viewpoint: which defines communities, actors, roles, activities and tools focused on rationales, opportunities and strategies.
- The information viewpoint: which defines the information handled by the various system resources and how it is dealt with by the various components.
- The computational viewpoint: which defines the software mechanisms that allow component diffusion and execution on software platforms.
- The engineering viewpoint: which focuses on the component mechanisms and functions.
- The technology viewpoint: which defines the material and software technologies.

The second abstraction level defined terms that allow us to specify the coherence between viewpoints. Four abstract concepts are used to specify our interpretation of e-learning platform project design practice:

- Decomposition: “To specify objects and behaviours which constitute an object or a behaviour” ([4], item 9.3) ;
- Template: “The specification of the common features of a collection of entities in sufficient detail that an entity can be instantiated using it” ([4], item 9.11) ;
- Behavioural compatibility: "An object is behaviourally compatible with a second object with respect to a set of criteria" ([4], item 9.4) ;
- Trace: “A record of an object’s interactions, from its initial state to some other state” ([4], item 9.6).

These concepts help us to describe four specification tasks. These consist of the analysis the source code, the design framework and the technical specification models applied in an e-learning system project.

3 Instance of RM-ODP framework on an e-learning system project: OpenUSS

By using the two abstraction levels proposed by RM-ODP framework, we explicit the design practices of OpenUSS¹ (Open University Support System). This open source community manages a web site which allows access to the e-learning system and diffuses the source code be installed, executed, copied, distributed, analysed and modified. The stability and the continuity of this community, the variety of roles and the number of the users (10,000 users: students, professors and assistants) justify our choice.

We create an instance of RM-ODP framework on this project:

- To abstract the design practices on EJB (Enterprise JavaBeans) component technology in e-learning ;
- To relate the informational models provided by the IMS consortium with an implementing entity ;
- To identify new perspectives of the ALT engineering.

3.1 Step 1 : Using the decomposition concept to specify the computational entities

This first step consists of specifying the platform's software architecture and describing the complex links between its components. We identify the first model published² on the OpenUSS web site. The dependence links use the UML (Unified Modelling Language) notation system describes the central position of the “foundation” component. This diagram corresponds to the first step of the abstraction process. However, this UML class diagram used describes the computational decomposition and the interactions between interfaces of different objects. This diagram corresponds to the computational viewpoint of RM-ODP framework. The analysis process must continue to transform the first specification into a more detailed specification.

The technical system, on which these components are deployed, is a platform compatibility with the EJB component technology of Sun Microsystems. Each component is specified by the deployment models provided by OpenUSS community.

In addition, the decomposition logic applied by the technical framework of this platform identifies the object type “workflow” as being central. Its associated model describes the states of the computational interfaces. This allows us to observe the behaviour by tracking the methods invoked on the object interfaces, and the operations of the container object in which it functions. Focused on this viewpoint, the analysis is limited.

¹ See, <http://openuss.sourceforge.net/>

² See, <http://openuss.cvs.sourceforge.net/openuss/openuss/design-general/model/src/>

This only communicates the production information to the technical administrator of platform. This information needs to be related to other aspects specified in the other viewpoints of the RM-ODP framework.

3.2 Step 2 : Using the template concept to specify the engineering choices

The decomposition of objects associated with the “foundation” component provided by OpenUSS community gives a list of objects from a computational viewpoint. The identification of the relations between two viewpoints allow us to transform a specification into a more detailed specification.

In this example, we look to model the relation which exists between the objects by specifying the computational objects from the engineering viewpoint.

The project OpenUSS designers apply a design process described in the technical framework EJOSA³ (Enterprise Java Open Source Application). It defines an easy to understand structure to be followed by the development with EJB components. This framework integrates a template which requires the naming rules for the source code files, and a structure in which to organize them.

Any designer who wants to collaborate with the OpenUSS community must respect this pattern when submitting a new component. The designer must save the structure of class in the “specification” file directory, produced by the analysis process. The UML class diagram used corresponds to the class saved in the “specification” file directory⁴ of “foundation” component. This UML diagram is the result of activity specification from an engineering viewpoint.

3.3 Step 3 : Using the behavioural compatibility concept to specify the conformance points

In addition, the RM-ODP framework defines itself as a guide to specify the conformance points between the ODP standard and a new specification. The act of specifying a compatibility between two objects bring us to define the relations between the specification selected and implementation the corresponding.

The conformance points depend on the terms of specification selected with observable behaviour in the e-learning platform. Moreover, for this framework the conformance points need to be identified and tests defined to satisfy in these points ([4], chapter 15). One of the four given conformance points classes defines “the behaviour of some physical medium so that information can be recorded on one system and

³ See, <http://www.enhydra.org/ejosa/>

⁴ See, <http://openuss.cvs.sourceforge.net/openuss/openuss/dev-foundation/specification/src/>

then physically transferred, directly or indirectly, to be used on another system” ([4], item 15.3.4).

The IMS Learning Design informational model [5] is the physical medium chosen. It is produced by a normalization process which aims to propose models that answer to the interoperability needs between e-learning systems. In addition, this model is used in the design practice of e-learning platform for discussion on the pedagogical practices [7] and on the learning scenarios [8].

3.4 Step 4 : Using the trace concept to specify the conformance tests

This last step seeks to validate the behavioural compatibility specification. Our choice used the IMS Learning Design informational model to represent the trace generated by the method invocation of “foundation” component.

Three schemas provided by RM-ODP framework guide the specification activities from the informational viewpoint: the static schema, the dynamic schema and the invariable schema ([3], chapter 6).

By taking the static aspect of the XML (eXtensible Markup Language) schema of IMS Learning Design informational model as the invariable schema, our interpretation is to consider that the behaviour of the “role” entity is linked to the behaviour of “activity structure” entity and “environment” entity.

In addition, the dynamic aspect identifies new informational objects : “method”, “play”, “act”, “role part”, “condition” and “property”. As specified in the IMS Learning Design specification document [5], these entities allow the description of the dynamic aspect of the scenario and correspond to the static schema from informational viewpoint.

4 Assessment of this analysis: New perspectives of ALT engineering

This article presents an instance of a framework on e-learning system project. We show the capabilities of the framework to support the designers team who wish to analyse the e-learning system design intentions.

The instance of the RM-ODP framework on the e-learning platform project demonstrates the four specification design steps. The effort of the OpenUSS community to communicate their e-learning platform design logics is highlighted in this example. The work of building relationships with new point of view proposed by another open source community is facilitated.

The aim is to understand and identify new perspectives on ALT engineering based on the abstraction of the design practices :

- on educational re-engineering which applies the propositions of ALT. The IMS consortium questioned its capacity to describe the *a priori* behaviour early on. One of the new perspectives is to analyse the capabilities of a generic framework like RM-ODP to instrument specification activity. For example in [8], the authors define rules to guide the observable specification based on and to extend the XML schema of IMS Learning Design language;
- on software based components engineering where each component can be reused to design another and to support the negotiation between them [1]. The integration a software tracker, itself constrained by software architecture, is a complex task. With the RM-ODP framework, we can analyse the designer practices on the OpenCPMS (open Controlling Performance Measurement System) component. It's a proposition of a tracking manager of the OpenUSS platform.
- on service engineering [11] which look to mapping the notion of business process and the business process life cycle onto learning processes and the learning process life cycle. This perspective structures and specifies the politics and the collective behaviours of a "community" defined in the business viewpoint [13]. These specifications will facilitate the exchanges between e-learning system projects;
- on model driven engineering where the difficulty is to identify the mechanisms used to model the various software artefacts [9]. The open source community OpenUSS [10] proposed one of the first contributions in the e-learning domain. One of the new perspectives is to detail this proposition by using RM-ODP framework concepts and viewpoints.

One of the stakes of the e-learning designers' open source community is to have a framework such as RM-ODP to propose a common vocabulary to share their practices and to identify new engineering and re-engineering perspectives to accommodate advances in ALT.

References

1. Szyperski. C (2003) Component Technology - What, Where, and How ? (USA). doi:10.1109/ICSE.2003.1201255
2. O'Rourke C, Fishman N; Selkow W (2003) Enterprise architecture using the Zachman framework. THOMSON LEARNING, Boston
3. ISO/IEC-10746-1 (1998) Open Distributed Processing Reference Model, Part 1: Overview. ISO/IEC JTC1 SC7
4. ISO/IEC-10746-2 (1996) Open Distributed Processing Reference Model, Part 2: Foundations. ISO/IEC JTC1 SC7
5. Koper R, Olivier B, Anderson T.(2003) IMS Learning Design Information Model, version 1.0 Final Specification. IMS Global Learning Consortium
6. Barré V, Choquet C (2005) Language Independent Rules for Suggesting and Formalizing Observed Uses In a Pedagogical Reengineering Context. doi: 10.1109/ICALT.2005.189
7. Burgos D, Koper R.(2005) Practical pedagogical uses of IMS Learning Design's Level B. SIGOSSEE Conference. <http://dSPACE.OU.NL/handle/1820/471>. Accessed 01 April 2008

8. Pemin J.-P., Lejeune A. (2006) Models for the re-use of learning scenarios. <http://dspace.ou.nl/handle/1820/580>. Accessed 01 April 2008
9. Bézivin J.(2006) Model Driven Engineering: An Emerging Technical Space. doi:10.1007/11877028
10. Grob H. L, Bensberg F., Dewanto B. L (2005) Model Driven Architecture (MDA): Integration and Model Reuse for Open Source eLearning Platforms. Eleed Journal. <http://eleed.campussource.de/archive/1/81/>. Accessed 01 April 2008
11. Helle D, Hrastnlk J, Maurer H (2005) An Analysis of Application of Business Process Management Technology in E-Learning Systems. Proceedings of E-Leam 2005 MCE Charlottesville USA 2937-2942
12. Cees V, C Jones V, van Sinderen et al (1997). Tele-education process modeling supported by the ODP viewpoint enterprise language. doi:10.1109/EDOC.1997.628341
13. ISO/IEC-15414 (2006) Open Distributed Processing Reference Model, Enterprise language. ISO/IEC JTC1 SC7

Assessing Innovation in the Software Sector: Proprietary vs. FOSS Production Mode. Preliminary Evidence from the Italian Case

Dario Lorenzi and Cristina Rossi
Department of Management, Economics, and Industrial Engineering
Politecnico di Milano

Abstract. Innovation in the software sector is a widely debated issue. Which are the most important dimensions to assess innovation in this field? Can we measure innovative processes carried out by software companies and what kind of innovation do they develop? Are FOSS solutions more innovative than proprietary ones? These are the research questions we endeavor to answer in this paper providing some empirical evidence, obtained using an original methodology of collecting experts' evaluations on the innovation level of 134 solutions provided by a group of Italian Small and Medium Enterprises.

Keywords: Innovation, Free/Open Source software, proprietary software

1 Introduction

The issue of innovation in the software industry is of great interest for both academic and practitioners (Bloch, 2007). Which is an innovative software solution? Which aspects should be considered to highlight the most important elements of innovation processes?

In this framework, the success of Free/Open Source Software (FOSS) highlights new research issues, dealing with whether and how FOSS programs turn out to be more innovative than proprietary ones (Klincewicz, 2005). The theme is intriguing as FOSS represents a disruptive process innovation affecting industrial dynamics (Dalle et al., 2007), and it can be regarded as an important instantiation of the Open Innovation model (West and Gallagher, 2006).

We focus on Italian software sector, addressing three research questions: (i) are software solutions developed by Italian SMEs' innovative? (ii) What kinds of innovations are implemented? (iii) Are FOSS programs more innovative than proprietary ones?

Please use the following format when citing this chapter:

Lorenzi, D. and Rossi, C., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succì; (Boston: Springer), pp. 325–331.

2 Review of Literature

The concept of innovation is one of the mostly studied by economic scholars (Fagerberg, 2004). Particularly, it was acknowledged the complexity of the phenomenon and how to measuring it turns out to be a critical task. A short list of the most used metrics of firms' innovation effort include: (i) data on Research and Development; (ii) patents; (iii) bibliometrical data, such as publications in scientific and technical journals.

All previous indicators suffer from several shortcomings (Kleinknecht et al., 2002), in particular as far as the sectors forming the so-called *new economy* (Haskel, 2007). Indeed, the passing from a commodity-driven to a knowledge-driven economy forces to re-define innovation, considering a whole new set of variables related to how knowledge is created, managed, and passed through different actors (Bloch, 2007). Specifically, it has been noted that patents are often unable to follow the rapid evolution of the software market and to account for the complexity of complementary activities (Blind et al., 2004).

As a consequence, new indicators are needed to assess innovation in ICT fields (Maruyama et al., 2007). In the case of software, such indicators should take into account not only general aspects, but also specific elements (as the use of certain programming languages and platforms and so on).

In this framework, the rapid pace of diffusion of FOSS represents another source of complexity. Klineciewicz (2005) has attempted to provide a classification of innovation in FOSS, basing on four classes: radical innovation, technological modifications, platform modifications, and new uses for existing technologies. The authors have also analysed 500 projects hosted on SourceForge¹, assessing that 436 of them are not innovative. This result can be linked to other studies showing low confidence in the innovative impact of FOSS solutions (Tuomi, 2005). In brief, the debate is far from a conclusion, even if FOSS solutions have proved to be as complex and reliable as proprietary ones.

3 Data and Methodology

The sample comes from a survey taken, in 2004, on more than 900 software firms² from Finland, Germany, Italy, Portugal, and Spain (Second European Libre Software Survey, ELISS II, see Bonaccorsi et al., 2006). Specifically, we focused on the 323 Italian respondents, because of database characteristics (some information are not available for all countries) and peculiarity of this software market, shaped by the presence of Small and Medium Enterprises (SMEs). More than 90% of firms have a total staff of less than 20 employees, and, in 16% of cases, they are one-man businesses. 167 of the companies provide to their customers FOSS-based software.

¹ <http://www.sourceforge.net>

² NACE code 72.2.

The average percentage of graduate staff is fairly high (about 36%), so as that of software developers (almost 30%). Respondents serve mainly business customers (81%), particularly SMEs (64%), while very few refer to University (3%) or end users (3%).

In order to address whether traditional innovation indicators are really not suitable for assessing innovation, data on three main metrics have been collected: (i) trademarks, using the database of the European Office for Trademarks and Design; (ii) patents, referring to the Delphion database (); (iii) scientific and technical publications hosted on Scopus³. Methodology is based on evaluations by software experts, basing on information recorded through the Web sites of the 323 Italian respondents. We collected comprehensive information on 134 solutions developed by a subsample of 70 companies. It is important to underline that, according to this methodology, the unit of analysis is no more the firm, but its software solutions. The 134 programs target mainly business customers: only 8% are intended for home users, and the most widespread category is managerial systems (45%), followed by Web oriented applications (9%). Moreover, most of them (107) are released under proprietary license, while the remaining ones (27) are distributed under FOSS licenses.

Three experts⁴ were asked to evaluate the level of innovativeness: after an in depth colloquium, they received a table (containing the name of each product, the link to its Web page, a brief description of the product and of the producing firm) to be filled and a guide for its compilation. It is worth noting that, in order not to introduce a bias, we did not tell the experts whether a software was FOSS or not; anyway, they might find this information on the software Web site. Evaluators were asked to assign a mark between 1 (*not innovative*) to 5 (*very innovative*) to each product, referring to three main dimensions, leading to a set of five indicators. The choice of the dimensions is driven by economic literature on the topic (see, for instance, Garcia and Calantone, 2002).

The first dimension is related to the internal level of innovation, comparing each software with other programs developed by the same company (*Indicator 1: is the product new to the firm?*). The second one refers to the market in which the firm operate, comparing each product with similar available solutions. This dimension is addressed by two indicators, related to innovation in *what* a software does (*Indicator 2: is the software innovative in the sense that it better satisfies needs from users?*), and in *how* it succeeds in accomplishing a task, exploring technical aspects (*Indicator 3: is the product technologically new to the market?*).

The third dimension addresses innovation referring to the general level of technology and knowledge (*is the product new to the world?*). Two indicators are developed: (i) innovation in modules composing the software (*Indicator 4: is the software innovative as it contains original modules, which can be hardly found in other solutions, also in different market segments?*); (ii) innovation in what we labelled as *platform*, meaning those aspects related to the other technological characteristics, such as

³ <http://oami.eu.int>, <http://www.delphion.com>, <http://www.scopus.com/scopus/home.url>

⁴Our decision to choose these experts was driven by geographic closeness and budget considerations. We are confident that they are well qualified to evaluate these products, having a long working experience in the software field.

programming languages, implemented algorithms, use of libraries, and so on (*Indicator 5: how the software platform is new with respect to those of other software solutions, also in different market segments?*).

It is useful to relate our indicators to dichotomy between *radical vs. incremental innovations*. For instance, innovations more related to technical aspects (indicator 3 and 5) can be considered as more radical innovations, with respect to a solution having original modules as the only element of novelty.

The main shortcoming of this methodology is the subjective nature of evaluations. However, all software have been evaluated by three experts: their knowledge of the market and multiple opinions mitigate the problem. Moreover, metrics of accordance of their judgement have been computed, showing that they were correlated⁵. Concluding, we chose to compute, for each indicator, the sum of the three evaluations, obtaining scores ranging between three (all experts assigned one) and fifteen (all five), and preserving variability even if mitigating the effect of outliers.

4 Empirical Results

Traditional metrics seem not to be suitable for the software sector formed. According to our investigations, only 5 companies registered, globally, 35 trademarks (26 logos and 9 names of companies). Only 3 firms hold a patent (for a total of 15 patents), and 3 were involved in scientific publications. In presenting experts' evaluations, we follow the repartition previously mentioned: innovation within the firm (dimension 1, indicator 1); innovation within the referring market (dimension 2, indicators 2 and 3); overall innovation (dimension 3, indicators 4 and 5).

Table 4.1: Descriptive statistics.

Dimension	Indicator	Min.	Max.	Std.Dev.	Median	75 th	90 th perc	99 th
I	1	4	14	2.11	8	10	11	14
	2	3	14	2.46	9	10.7	12.7	13.7
II	3	3	14	2.78	8	10	12	13.7
	4	5	13	2.04	8	10	11	13
III	5	3	14	2.46	8	9	11	13.7

For the first dimension, 99th percentile is 14 (the highest for all dimensions). Standard deviation is low, showing concentration around mean values (more than 50% of the sample is between 6 and 8). Two indicators (innovation in *what* a software does: *indicator 2*; innovation in technical aspects: *indicator 3*) capture the second dimension. Both show equal values for minimum and mode, while differences in percentiles emerge: in 75th and 90th percentiles, indicator 2 has higher values (respec-

⁵ As our study is characterised by multiple ratings per subject, we used the Spearman's rank correlation coefficient.

tively, 10.7 vs. 10, and 12.7 vs. 12). This finding is confirmed by other analysis: 69% of solutions have received a mark of 7 or more for indicator 2, while the proportion decreases to 52% for the third indicator. The innovative process seems to be more effective for aspects related to what a software does, instead of technological aspects. Third dimension was distinguished between innovation in modules (*indicator 4*) and in other technical aspects (*indicator 5*). 75th is higher for indicator 4 (10 vs. 9), but this pattern is not still true when focusing on values closer to the upper limit (99th percentile: 13 vs. 13.7). Experts claimed that sometimes Web sites provided few information about modules: this have generated a *central tendency bias*, leading evaluators to assign score 3 (confirmed by standard deviation: 2.04 for modules vs. 2.46). Indicator 4 obtains higher scores: proportion of evaluations between 5 and 7 is predominant for indicator 5 (32% vs. 38%), while indicator on modules presents more values between 7 and 9 (53% vs. 43%). This may be related to the peculiar structure of the sample: in a market dominated by SMEs, it is likely to observe more customizations and adaptations than radical innovations based on new technologies.

Summing up, data allow us to answer the first two research questions. As far as the former (*are Italian software solutions innovative?*), whilst results with traditional instruments are useless, using a methodology based on experts' evaluations, we succeed in disentangling innovation into its main dimensions. Also the latter issue (*what typologies of innovation are implemented?*) receives an answer. On one hand, when focusing on the referring market, indicator 2 shows higher values than indicator 3, supporting the idea of innovation processes more targeted to *what* a software solution does, than to *how* to do it. On the other, considering the *new to the world* dimension, innovation seems to be more focused on modules than in technical aspects. According to our findings, the Italian software sector seems to be characterized mainly by adaptations, customizations, and transfer of solutions into different platforms (an innovation that can be appointed as incremental).

In order to address our third research question (*are FOSS solutions more innovative than proprietary ones?*), we have performed comparisons between these two groups. As mentioned in the previous section, the sample of 134 software was formed by 109 proprietary and 27 FOSS solutions.

Table 4.2: Descriptive statistics. Proprietary vs. FOSS solutions

		Proprietary solutions							FOSS solutions						Mann Whitney P value	Nonparametric equality of medians P value
Di m	Ind	Mi n	Ma x	Me-dian	Std Dev	75 th per	90 th per	Mi n	Max	Me-dian	Std Dev	75 th per	90 th per			
I	1	4	14	8	2.03	10	11	6	14	10	2.15	11	12	0.007	0.027	
	2	3	14	8	2.30	10	11	5	14	11	2.66	12.5	13	0.004	0.002	
II	3	3	14	7	2.55	9	11	3	14	11	3.14	12	13	0.003	0.001	
	4	5	13	8	2.03	10	11	5	13	9	2.12	10	11	0.428	0.061	
III	5	3	14	7	2.21	9	10	4	14	10	2.58	11	12.4	0.000	0.000	

FOSS solutions seem to be more innovative: in all the three dimensions, evaluations are higher for FOSS. Specifically, indicator 1 has higher median for FOSS and statistical tests show that this difference is significant. This is corroborated by data distribution: focusing on the score range 5-7, the proportion for FOSS (26%) is lower than the one for proprietary programs (44%), while the opposite emerges for values equal to 10 or more (51% vs. 27%). As far as the second dimension, FOSS solutions seem to be more innovative too. For both indicators, statistical tests confirm a higher level of innovativeness. Focusing on differences between indicators 2 and 3 within the same group, we notice that, in both cases, evaluations for indicator 3 are lower than those of indicator 2, confirming the conclusions for the entire sample. Same pattern emerges with the third dimension. However, whilst statistical tests confirm significant differences in median for indicator 5, no significant difference between the two sub-samples emerges for indicator 4, but, as mentioned above, results could be invalidated by central tendency bias. If the comparison between the two groups is interesting, it is also of interest to consider the differences between the two indicators within each group. Indeed, proprietary solutions show higher evaluations for modules, as emerged for the entire sample (e.g. values over 7 constitute the 65% for indicator 4, while only the 46% for indicator 5), while FOSS programs follow an opposite pattern. Indeed, indicator 5 shows higher values, as highlighted by median (9 for indicator 4 vs. 10 for indicator 5), 75th percentile (10 vs. 11), and 90th percentile (11 vs. 12.4). A more in depth analysis allows to notice that a large part for this difference comes from the highest values. This last consideration can be regarded as an insight that proprietary and FOSS software are also shaped by different innovative processes: radical innovation in FOSS vs. incremental innovation in proprietary field.

5 Conclusions

Innovation in software industry can be hardly defined and measured: this paper proposes an original methodology for assessing innovativeness of software solutions produced by Italian SMEs. We have provided evidence that innovative processes exist and it is possible to assess them under different perspectives and to delineate general characteristics of these processes, showing that innovations related to technical aspects are less prominent: it is possible to address these innovation processes as incremental, which is in agreement with a software sector shaped by SMEs. Moreover, our analysis has highlighted differences between innovativeness of FOSS and proprietary software. Specifically, FOSS solutions display higher level for all indicators, and almost all statistical tests indicate that the two groups are two different populations. Differences emerge in the level of innovation and in the relationships between indicators: specifically, focusing on the global level of technology, FOSS solutions show higher values for technical aspects. The characteristics of the sample made results hardly generalizable; however, they help in shed light on questions posed by the FOSS paradigm. Does the FOSS production foster innovation processes? Does the FOSS represent a valid alternative for software SMEs wishing to operate with lead-

ing-edge technologies? Hence, it would be interesting for future researches to apply a refined version of this methodology on larger database, containing data from different countries.

References

- Blind K., Edler J., Friedewald M. (2004) *Software Patents – An Empirical Analysis from an Economic Perspective*. Stuttgart: Fraunhofer IRB Verlag
- Bloch C. (2007) Assessing recent developments in innovation measurement: the third edition of the Oslo manual. *Science and Public Policy*, 34 (1): 23 – 34
- Bonaccorsi A., Giannangeli S., Rossi C. (2006) Adaptive entry strategies under competing standards-hybrid business models in the Open Source software industry. *Management Science*, 52(7): 1085-1098
- Dalle J.M., Rossi C., Rullani F. (2007) Toward a new industrial organization? OSS in economic and managerial perspective. In Feller J., Fitzgerald B., Scacchi W., Sillitti A. (eds.) *Open Source Development, Adoption and Innovation*, Springer, New York, NY
- Fagerberg J. (2004) Innovation: a guide to the literature. In Fagerberg, Jan, David C. Mowery and Richard R. Nelson: *The Oxford Handbook of Innovations*. Oxford University Press: 1-26
- Garcia R., Calantone R. (2002) A critical look at technological innovation typology and innovativeness terminology: a literature review. *he journal of product innovation management*, 19(2): 110-132
- Haskel J. (2007) Measuring innovation and productivity in a knowledge-based service economy. *Economic and Labour Market Review*, 1 (7): 27-31
- Kleinknecht A., van Montfort K., Brouwer E. (2002) The Non-Trivial Choice between Innovation Indicators. *Economics of Innovation and New Technology*, 11 (2): 109-121
- Klincewicz K. (2005) Innovativeness of open source software projects, MIT online paper, Boston. Available at: <http://opensource.mit.edu/papers/klincewicz.pdf>
- Koberg S., Detienne D., Heppard K. (2003) An empirical test of environmental, organizational, and process factors affecting incremental and radical innovation. *The Journal of High Technology Management Research*, 14 (1): 21-45
- Maruyama, Kohda, Katsuyama (2007) Issues of Service Innovation and Its Model. The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services: 489-490
- West J., Gallagher S. (2006) Patterns of open innovation in Open Source software. In Chesbrough H., Vanhaverbeke W., West J., Eds. (2006) *Open Innovation: researching a new Paradigm*, Oxford University press, Cambridge

Detecting Agility of Open Source Projects Through Developer Engagement

Paul J. Adams¹, Andrea Capiluppi², and Adriaan de Groot¹

¹ Sirius Corporaion Ltd., Hamm Moor Lane, Weybridge, UK, KT15 2SF
paul.adams@siriussit.co.uk, groot@kde.org

² Centre for Research on Open Source Software, University of Lincoln, Lincoln, UK, LN5
7TS acapiluppi@lincoln.ac.uk

Abstract. The principles behind the agile development methods and common practise within the Open Source community are vastly different. In recent years there has been a rise of interest in these, in order to detect and inform on areas of compatible shared practises. This paper argues that it is possible to quantify the level of agility displayed by Open Source projects. An indicator of agility, the Mean Developer Engagement (MDE) metric is introduced and tested through the analysis of public project data. Projects sampled from two repositories (KDE and SourceForge) are studied and a hypothesis is formulated: projects from the two samples display a similar level of MDE.

This paper provides two main contributions: first, the MDE metric is shown to vary significantly between the KDE and SourceForge projects. Second, by combining MDE with a project's lifespan, it is also shown that SourceForge projects have insufficient uptake of new developers resulting in more active, shorter, initial activity, and in a quicker “burning out” of the projects.

1 Introduction

Plan-driven approaches to software engineering emphasise large-scale planning and formal communications while agile approaches emphasise flexibility [14]. Proponents of agile processes argue that such processes respond to change and withstand the continual pressures of software evolution better than plan-driven approaches [6, 1]. Proponents of the OSS paradigm claim that it should be considered as a novel approach, and its evolution and maintenance phases should be considered different from a plan-driven approach [5].

In theory agility and Open Source are very different concepts; the latter being just a licensing paradigm with implications for code reuse and redistribution. Comparative studies have been made in the past, but the scarcity of data from Agile processes leaves most studies on the surface of theoretical discussions [19, 13].

The term “agility” has been formulated to assess how a software company locates itself in a spectrum between Agile and plan-driven. Process-related characteristics have been identified to help either in focusing on the critical factors of the development framework [4]; or in the selection itself of the most appropriate approach [7]. Agility in Open Source projects, however, is often implemented significantly differently from the canonical definition of the agile processes. Usually this occurs due to certain agile

Please use the following format when citing this chapter:

Adams, P.J., Capiluppi, A. and de Groot, A., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 333–341.

practises requiring team co-location [15] or smaller teams [12]. Instead agility within Open Source is often performed by much larger teams in a distributed fashion; although co-location for certain activities, like sprinting, is becoming more common [9]. The unifying aspect of all agile processes is flexibility in order to react to change (in requirements, or other conditions) [3]. We posit that this, coupled with the agile culture of producing useful, working software as quickly as possible, creates an implicit requirement for an agile project to make efficient use of its developer resource.

This paper introduces and justifies the Mean Developer Engagement (MDE) metric. Being it a process-based metric, it correlates with the effort of OSS developers along the duration of a project, and it addresses one of the relevant factors in the definition of agility [4]. The metric is not introduced per-se: it is used both to empirically evaluate the agility of OSS projects, and to compare two samples of projects extracted from two repositories (KDE and SourceForge), in order to detect differences in the agility of OSS developers. The traditional Goal-Question-Metric paradigm will be used to detect the presence of differences: a research hypothesis will be formulated and a statistical test will be evaluated to detect statistically significant differences among the samples.

This paper is structured as follows: Section 2 introduces the MDE metric and describes the factors it is built upon, showing how the data was gathered and processed. Section 3 introduces the GQM approach and tailors it into the research hypothesis which this paper explores. Section 4 evaluates the results and provides the discussion of the statistical tests used. Section 5 concludes.

2 Mean Developer Engagement

Users and developers are limited resources within the OSS environment, particularly so with regards to developers [8]. However, several major OSS projects have been noted to be particularly successful at recruiting new developers [16, 17]. No empirical evidence has been given yet to a comparison of how repositories employ their resources, in terms of active developers.

In order to draw a comparison between two OSS repositories, the Mean Developer Engagement (MDE) is here proposed and analysed as a metric. MDE is a measurement of how effective (on average over time) an OSS project is at making use of its human resources and, therefore, potentially indicative of agility [7]. MDE is measured over n time periods, and each developer associated with the project is classified as active or inactive in each period. Typically, a week is used as time period; the number n ranges from 1 (a very short project) to over 600 (long-term projects). In our research we define MDE as:

$$\bar{d}e_n = \frac{(n-1) \cdot \bar{d}e_{n-1} + \left(\frac{\text{dev}(\text{active})}{\text{dev}(\text{total})} \right)_i}{n} \quad (1)$$

– $\text{dev}(\text{active})$ is the number of (distinct) developers active in time period i .

- $dev(total)$ is the total number of developers involved with the project in the periods $0 \dots i$.
- n is the number of time periods over which the project has been evaluated.

An ideal project, in which every developer is active in every single time period within the range of the MDE calculation, has an MDE of 1. The primary concern with the current definition of MDE lies with the metric $dev(total)$, this metric naïvely asserts that the number of accounts in the project repository is the total number of developers in the project. This is deficient as an account may remain in the repository long after a developer has left a project. To create a refined and more precise definition, $dtotal$ is augmented with the inclusion of a *grace period*.

2.1 Refined Definition – Grace Period

OSS project developers leave a trace which can be retrieved by assessing version control logs (*i.e.* their account name appears for the first time). The definition of $dev(total)$, above, establishes a list of everyone who has ever been a developer in a specific project. In order to detect when a developer leaves a project, it is possible to set an arbitrary time span after which a developer who has shown no activity in that span is removed from the list of project participants.

The definition of $dev(total)$, for each n , should be refined in order to detect developer inactivity. Depending on length of service, each developer shall be allowed an inactivity “grace period,” as are shown in Table 1 (These figures were created as a discussion exercise between 10 Open Source developers and validated by 10 more). Only once they have been inactive for longer than this time shall they no-longer be considered as project developers. As soon as the developer commits more code, they are reconsidered as a developer and their grace period is set to one, as a new developer.

Table 1. Developer grace periods (in weeks), shorter length of service has grace period 1.

Length Of Service	Grace Period	Length Of Service	Grace Period
1024	20	208	8
520	15	104	6
416	12	52	4
260	10	24	2

By making these adjustments a more accurate measure of $dev(total)$ is calculated, thus improving the overall accuracy of MDE. The downside to this adjustment, however, is an increase in overall computational complexity when automating MDE measurement. This is simply a linear increase, with n , in complexity created by the adjust being made after each time quanta.

2.2 Data Gathering and Processing

In order to calculate MDE for a project we require to know which members of the project are active and when. In this research the data has been gathered from the history of the projects' source code repositories. This data could easily be supplemented with data from mailing lists, IRC logs, etc. The history contains, for each change done to the project, at least the user name of the developer and the date/time stamp of the commit. MDE can easily be calculated using these. Most source code management tools (such as Subversion) provide a way to print out the histories in a machine-parseable form.

Figure 2.2 shows example plots of MDE (in both simple and refined forms) for the entire KDE project (<http://www.kde.org>). KDE is a very large project, with over 1600 accounts in its Subversion repository, over 760000 changes and 10 years of development history. The plot shows the change in MDE as calculated on a week-by-week basis from the beginning of the project in 1997.

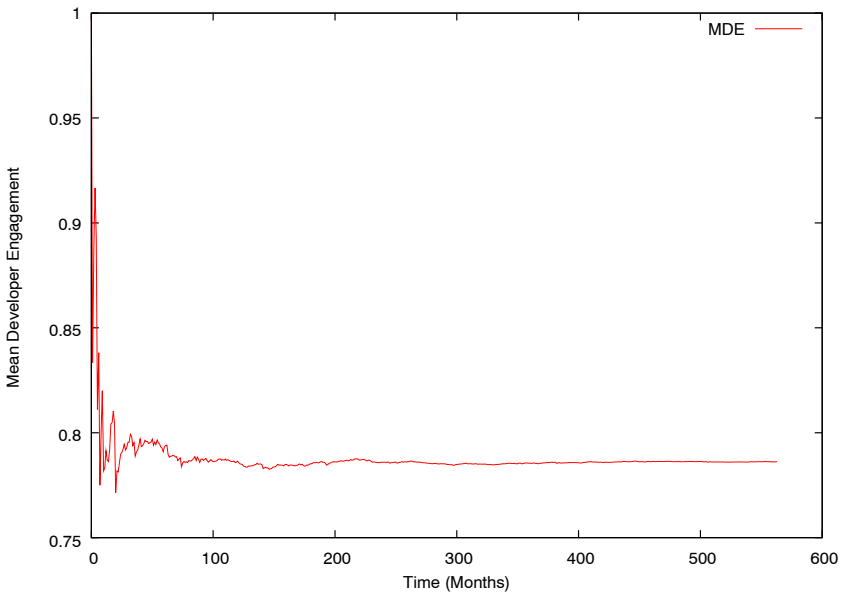


Fig. 1. MDE over the lifetime of the KDE project

The first aspect of MDE plots that should be noted is the anomalies which occurs within the data for the first year. These deviations from the trends are caused by a disproportionate shift in the number of active developers early on. Early in a project's

lifetime, both n and $\text{dev}(\text{total})$ are low and any change in $\text{dev}(\text{active})$ will have a dramatic affect on MDE.

The initial period of upheaval is followed by an upturn in MDE, indicating that the project has attracted many new developers. This is shortly followed by a slightly steeper gradient in the plot, indicating increased turnover (i.e. an increased number of developers having short tenure in the project). As the project matures, this plot begins to level out as more developers maintain longer commitments to KDE. From week 300 until the end of the plot the gradient of MDE is stable. This is indicative of process (explicit or otherwise) and a stable community.

By applying the adjustments described by Table 1 the new plot is always equal to or greater than the original plot at any point in time. This is clearly shown in Figure 2.2 where the refined MDE plot is significantly higher in value. In the case of this KDE plot, the most remarkable aspect is the stability and success with which KDE has managed to engage its active developer base; this plot clearly indicating that the average regular contributor has been active nearly 80% of the time since the second year of the project.

3 MDE – Empirical Evaluation

In order to validate the MDE metric, the following Section will both apply it and study its behaviour on two samples of OSS projects: one was extracted from KDE, and the other was randomly selected from the repositories hosted at SourceForge (<http://sourceforge.net>). SourceForge (SF) hosts over 120,000 different projects; the majority of these are small projects compared with a large and long-term project like KDE (which is outside SourceForge) [10].

The MDE metric applied to the KDE project as a whole is analysed in Figure 2.2 above. The KDE project has to be considered as a collection of sub-projects, each of them much more focused on a particular goal, and with a smaller developer base. Very few of the developers in KDE are active across the entire project, as most focus on their own sub-project. This improves the accuracy of MDE for each sub-project by disregarding developers who work elsewhere in KDE.

As the data-base for the present work, 20 sub-projects from KDE and 20 SF projects were randomly sampled. Projects which had no history, (i.e. “dead on arrival” as they have not ever engaged developers) were replaced by other randomly selected projects. A research hypothesis was formulated based on the MDE behaviour on these two samples. It is tested first via a bi-directional test, to detect major differences between the two samples; then a directional test is applied to detect whether one sample statistically achieves better results in terms of the MDE metric.

3.1 Empirical Approach – Goal-Question-Metric

Metric-based evaluation of software products is well established within the software engineering research, since it allows a simple and effective means of quality evaluation [11]. The Goal-Question-Metric (GQM) approach [2, 18] is particularly popular

due to its ease of automation. This research was conducted using the GQM approach, with MDE as a metric.

Goal: The goal of this research is to identify, through the application of metrics, the degree of agility displayed by an Open Source development project. Empirically comparing the Agile and OSS development approaches, finding similarities and differences, will help bridge the gap between the traditional closed-source and newer paradigms in software development.

Question: As part of a greater research programme this study is conducted to establish the statistical significance of MDE as a metric. As such, we ask: Are projects within KDE statistically different from those hosted within SourceForge with regards to MDE? Alternatively, does KDE display a different level of agility than SF projects? If a difference is found in the bi-directional comparison of the samples, a more restrictive test will be formulated, asking whether the KDE sample consistently achieves better MDE results than the SourceForge sample.

Metric: The metric applied is MDE using time quanta of one week. The metric is applied across the project lifetime and an average MDE score for each week is used.

3.2 The Samples

The complete version control log of the 20 randomly selected KDE projects and SourceForge projects were parsed to produce the MDE scores shown in Table 2. Here, the date of the first log entry, i (the duration of the log in weeks) and the MDE score are shown for each project. In addition a “effort” value is included and will be described in the following section.

4 Results and Analysis

From the SourceForge sample, one may note the relatively high incidence of projects with an MDE of 1.0, indicating “ideal” projects with total engagement of all the project developers. A closer examination of the table shows a strong correlation between an MDE of 1.0 and a lifespan of one week; a project *necessarily* has an MDE of 1.0 in its first week, as the determination of active and total developers is based on the set of developers seen in that first week. Such short-lived projects are likely “dead on arrival” as they fail to engage any long-term effort [10]. Much more interesting are the longer-lived projects with a high MDE, such as wxpropgrid (MDE=0.9968 over seven months) and shareaza (MDE=0.7830 over three years).

4.1 Testing of the Hypothesis and Discussion

The Wilcoxon two-sample test is used to assess the significance in similarity between samples from two populations. It is applied here principally in two-tail form to show that the samples are indeed significantly different and yields: $W = 97$, $p \leq 0.005376$. At the 95% confidence level, this is a significant result. We reject the null hypothesis that the populations display similar levels of agility. We test the hypothesis that SF

Table 2. MDE Scores Calculated for 20 Random Projects from KDE and SF.net.

KDE					SF.net				
Project	Start	<i>i</i>	MDE	Effort	Project	Start	<i>i</i>	MDE	Effort
dolphin	21-11-06	54	0.6799	36.7146	askcms	29-06-06	1	1.0	1.0
k3b	26-03-01	349	0.5961	208.0389	awdotnet	24-05-07	1	1.0	1.0
katomic	29-06-99	438	0.3340	146.2920	dvdshop	31-03-06	1	1.0	1.0
kcalc	13-04-97	554	0.4307	238.6078	hivex	16-07-07	1	1.0	1.0
kgeography	07-03-04	38	0.5386	20.4668	interaction	04-03-07	1	1.0	1.0
kig	15-04-02	294	0.6878	202.2132	kuragari	13-01-07	1	1.0	1.0
kivio	02-12-00	365	0.5320	194.1800	kyrios	03-07-06	74	0.5634	41.6916
kmail	18-01-03	254	0.6730	170.9420	map	06-11-05	50	0.3780	18.9000
kmoon	27-09-98	478	0.2913	139.2414	neuralbattle	14-06-06	74	0.6803	50.3422
knotes	30-06-97	544	0.4638	252.3072	opulus	25-07-07	10	0.4931	4.9310
kolourpaint	10-10-03	214	0.6269	134.1566	pwytter	09-07-07	1	1.0	1.0
konqueror	09-02-99	459	0.6610	303.3990	pyaws	11-04-06	56	0.2514	14.0784
konsole	28-10-98	474	0.6109	281.5666	rejuce	02-08-06	10	0.5554	5.5540
kontakt	18-01-03	254	0.5867	149.0218	rlycyber	02-07-06	17	0.7612	12.9404
kopete	02-01-02	308	0.7142	219.9736	shareaza	02-06-04	182	0.7830	142.5060
kscd	04-07-97	542	0.4962	268.9404	stellarium	12-07-02	280	0.6183	173.1240
kspread	18-04-98	502	0.6216	312.0432	tblshp	04-10-06	1	1.0	1.0
ksudoku	03-03-07	39	0.6530	25.4670	tsg	23-03-07	19	0.6036	11.4684
kteatime	16-04-99	450	0.3299	148.4550	wxpropgrid	16-04-07	31	0.9968	30.9008
marble	29-09-06	61	0.6321	38.5581	xml-copy-editor	16-08-07	14	0.7731	10.8234

shows greater level of agility than KDE using a single-tail Wilcoxon test and find: $W = 97, p \leq 0.002688$. This is another strong result at the 95% confidence level, and we reject the null hypothesis. SF *does* show greater agility.

The Wilcoxon two-sample tests applied here strongly indicate a statistical difference between MDE in the samples from KDE and SF and potentially a difference in agility. We may say that the SF projects achieve better developer engagement in the sampled projects. It may be surprising that both null hypotheses were rejected, especially given SourceForge’s notoriety for being home to low quality projects [10].

This counter-intuitive result was studied further: an analysis of the start and end dates in the projects from both samples shows that KDE projects may not be as strong at utilising developers, but typically have longer lifespans. There may be further significance to MDE in conjunction with the lifespan of a project. Using the same samples from KDE and SF, the duration of the project was multiplied by the MDE value in order to produce a new “effort” score. The weighted effort value for each project is shown within Table 2.

Even without formal statistical analysis, we see a clear separation between the new “weighted” scores for MDE. The same two-tail Wilcoxon test applied to effort returns: $W = 374, p \leq 2.455e-06$. Therefore, at the 95% confidence level we can state that a randomly selected KDE project will display better developer engagement over time than a randomly selected SourceForge project. This can be explained as follows: SF projects tend to better engage developers, but just for a limited amount of time, and its projects tend to “burn bright” but fade rather quickly. The KDE project manages in-

stead to achieve a prolonged engagement of its developers, and this by itself represents a quality factor within different OSS repositories.

5 Conclusions and Further Work

This paper argued that it is possible to measure the agility of OSS projects by means of a metric based on developers effort. The Mean Developer Engagement (MDE) metric was introduced as a means for assessing how affectively OSS projects utilise their developer resource, if they leverage it consistently and for prolonged periods of time, and whether different OSS repositories achieve different levels of engagement of developers. Samples from two OSS repositories (KDE and SourceForge) were extracted, and the MDE's statistical significance was evaluated through a bi-directional and a single directional tests.

The empirical evaluation of the hypothesis showed at first a counter-intuitive result: if fact, it was found that SF projects are statistically better at engaging their developers than the projects from KDE. Investigating the results further, the duration of development was studied, which displayed much longer development periods in the KDE sample than in the SF one. Combining the duration with the engagement, we find that SF projects tend to better engage their developers, but only for a limited amount of time, after which it is common for the project to quit its activity. KDE projects have more overall endurance: part of the endurance beyond the lifespan of a SF project must be attributed to the more regular intake of new developers these projects have. Therefore a project is more likely to sustain MDE in a forge of related projects, such as KDE, than in a forge of unrelated projects, such as SourceForge.

Before MDE can be formally introduced as a metric further work shall be carried out to compare the results of Open Source MDE, presented here, with known agile project data from industry. This will allow us to establish upper and lower thresholds for MDE as an indicator of agility.

6 Acknowledgement

This work is partially supported by the European Community Framework Programme 6, Information Society Technologies key action, contract number IST-033331 ("SQO-OSS").

References

1. A. Capiluppi and J. Fernandez-Ramil and J. Higman and H. C. Sharp and N. Smith. An Empirical Study of the Evolution of an Agile-Developed Software System. In *International Conf. on Software Engineering*, pages 511–518, Minneapolis, Minnesota, May 2007.
2. V. Basili, G. Caldiera, and H. D. Rombach. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*, pages 528 – 532. John Wiley & Sons, Inc., 1994.

3. K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change*. Addison Wesley, 2004.
4. B. Boehm and R. Turner. Using Risk to Balance Agile and Plan-Driven Methods. *IEEE Computer*, 36(6):57–66, 2003.
5. A. Capiluppi, J. Gonzales-Barahona, I. Herraiz, and G. Robles. Adapting the “Staged Model for Software Evolution” to Free/Libre/Open Source Software. In *Proc. Int’l Workshop on Principles of Software Evolution (IWPSE)*, Dubrovnik, Croatia, 3–4 Sept. 2007.
6. N. Chapin. Agile Methods’ Contributions in Software Evolution. In *ICSM ’04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, page 522, Washington, DC, USA, 2004. IEEE Computer Society.
7. S. Datta. Agility Measurement Index: A Metric for the Crossroads of Software Development Methodologies. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 271–273, New York, NY, USA, 2006. ACM.
8. A. de Groot, S. Kügler, P. J. Adams, and G. Gousios. Call for Quality: Open Source Quality Observation. In E. Damiani and B. Fitzgerald and W. Scacchi and G. Scotto, editor, *IFIP International Federation for Information Processing: Open Source Systems*, pages 57 – 62, 2006.
9. B. Düring. *Extreme Programming and Agile Processes in Software Engineering*, chapter Sprint Driven Development: Agile Methodologies in a Distributed Open Source Project. Springer, 2006.
10. R. English and C. M. Schweik. Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of Sourceforge.net Projects. In *International Workshop on Emerging Trends in FLOSS Research and Development*, 2007.
11. Norman E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, Ltd., London, UK, 1991.
12. K. Schwaber and M. Beedle. *Agile Software Development with SCRUM*. Prentice Hall, 2002.
13. S. Koch. Agile Principles and Open Source Software Development: A Theoretical and Empirical Discussion. In *Extreme Programming and Agile Processes in Software Engineering: Proceedings the 5th International Conference XP 2004*, number 3092 in Lecture Notes in Computer Science (LNCS), pages 85–93. Springer Verlag, 2004.
14. M. Fowler and J. Highsmith. The Agile Manifesto. In *Software Development, Issue on Agile Methodologies*, <http://www.sdmagazine.com>, last accessed on March 8th, 2006, August 2001.
15. M. Kircher and P. Jain and A. Corsaro and D. L. Levine. *Extreme Programming Perspectives*, chapter Distributed Extreme Programming. Pearson Education, 2002.
16. A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.
17. G. Robles and J. M. Gonzalez-Barahona. Contributor Turnover in Libre Software Projects. In Ernesto Damiani, Brian Fitzgerald, Walt Scacchi, Marco Scotto, and Giancarlo Succi, editors, *OSS*, volume 203 of *IFIP*, pages 273–286. Springer, 2006.
18. R. van Solingen. *The Goal/Question/Metric Method: A Practical Guide For Quality Improvement Of Software Development*. McGraw-Hill, 1999.
19. J. Warsta and P. Abrahamsson. Is Open Source Software Development Essentially an Agile Method? In *3rd Workshop on Open Source Software Engineering*, 2003.

Facilitating Social Network Studies of FLOSS using the OSSNetwork Environment

Marco A. Balieiro, Samuel F. de Sousa Júnior, and Cleidson R. B. de Souza
Faculdade de Computação – Universidade Federal do Pará
66075-110 – Belém – PA – Brazil,
{ma.balieiro, sfelixjr}@gmail.com, cdesouza@ufpa.br

Abstract. Open source projects are typical examples of successful distributed software development projects. Understanding how coordination in these projects takes place can provide important lessons to Software Engineering researchers and practitioners. This understanding has been achieved using different research methods, including, surveys, case studies and social network analysis. However, to conduct these studies each researcher needs to build his own infra-structure from the scratch, a time consuming and error-prone task. This paper aims to alleviate this problem by describing an environment, the OSSNetwork, which allows the automatic data collection of open source repositories. Data collected by the OSSNetwork is aimed to support the construction, visualization, and analysis of social networks. This environment is extensible, therefore facilitating empirical studies of open source projects.

1 Introduction

The Free/Libre Open Source Software (FLOSS) movement has become an economically viable and financially satisfactory alternative to proprietary software due to its reduced cost, good performance in critical operations and data manipulation, and improved security [1]. Supporters of FLOSS argue that the availability of the source code influences positively its quality, since any developer can review the code and improve it [2]. Today, the Internet infrastructure has great part of its critical elements based on FLOSS [3].

Due to these factors, researchers and practitioners of several areas (software engineering, economy, sociology, etc) are interested in understanding FLOSS projects from different viewpoints, including developers' motivation [4], the software process adopted by these communities [5], quality assurance [6], just to name a few. To address these different goals, several research methods have been used, from ethnography [7], to case studies [8], social network analysis [9], and even traditional statistical methods [10]. In particular, the use of the Social Network Analysis (SNA) allows the study of relationships among developers in these communities. For instance, Lopez-Fernandez, *et al.* [11] used social networks to understand the social relationships based on data available in CVS repositories, whereas Crowston and Howison [12] studied the networks of people who got

Please use the following format when citing this chapter:

Balieiro, M.A., Júnior, S.F. and Souza, C.R.B., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 343–350.

involved in bug fixing activities, and de Souza, *et al.* [13] analyzed developers' social network extracted from source-code dependency relationships.

All these approaches helped to understand important aspects of FLOSS projects. However, a limitation of these approaches is that they were conducted independently: each researcher had to build his own tools to extract, manipulate and analyze the data. To minimize this effort, Howison, *et al.* [14] developed the FLOSSmole, a tool for collect, store and distribute FLOSS data and analysis. While FLOSSMole alleviates part of this problem, researchers still face difficulties to collect data from FLOSS repositories to perform social network analysis such as: collection and storage the data which are in servers that researchers do not have direct access, differences between data models from different repositories, handling large amounts of information, treatment of specific information that can generate ambiguities (such as the problem of aliases [15]), implementation of algorithms on the data collected or even on the social networks generated, and visualization and manipulation of these networks.

The work described in this paper extends the FLOSSmole tool with an environment, called OSSNetwork, which allows the study of FLOSS communities using social network analysis. This environment is extensible so that new algorithms, visualization, and functionalities can be added. Social networks are generated with information extracted from mailing lists, forums, issue-tracking tools and chat logs. All this information is stored in a local database for future analysis. We argue that the problems faced by SNA researchers described in the previous paragraph are minimized by using the OSSNetwork environment.

The rest of this paper is organized as follows. In the next Section we will briefly present our motivation to this work. Next, a very short review of social network definition will be presented. The following section describes the OSSNetwork environment, and is followed by a case study and a discussion of the obtained results. Finally, we present our conclusions and future work.

2 Motivation

Howison, *et al.* [14] describes a research process to be used when studying FLOSS projects. This process is presented in Figure 1. According to this process, one of the most difficult tasks is the selection of projects to analyze. According to Howinson, two mechanisms can be used: *census*, where all existing projects of a particular phenomenon to be examined must be used, which is very difficult because no one knows how many projects exists; and *sampling*, where a small number of projects that represents a particular phenomenon is selected randomly. This is not an easy task, because researchers actually end up limiting their studies to a single repository. More importantly, sampling open source projects is methodologically difficult because everything FLOSS research has shown so far points to massively skewed distributions across almost all points of research interest [16][17] and even

randomly selecting the projects, still, in general, does not produce a representative sample [14].

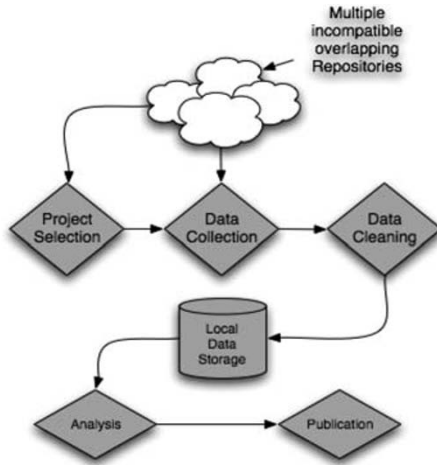


Figure 1 – Research process in FLOSS projects

Project selection becomes even more complicated because public repositories contain a large number of inactive, relocated or disabled projects. Our approach to handle this problem, detailed later, is to expand the number of supported repositories. After selecting the projects, two processes can be used for data collection: getting the databases *dumps* of the repositories or conduct a *parsing* of the repositories’ web pages. Although gaining access to the databases is clearly preferable, not all repositories make their dumps available. Therefore, parsing is the data collection method that is made available by OSSNetwork, because it only requires the availability of Internet access.

3 A brief Social Network overview

A social network is broadly defined as a set of relationships [18]. A social network has a set of objects (nodes, in mathematical terms) and a description of the relationships between these objects. For example, it can be said that nodes are the people of a house and the relation that establish connection between these people is “people who use the same room”.

A social network can be characterized according to structural and topological properties [19]. These properties are derived from the graph theory. The structural properties are: node degree, weighted degree of a node, distance centrality of the node, proximity degree, betweenness centrality, and others. Topological properties include: density of the network, distribution degree, network diameter, and finally,

cluster degree, that it is a set of connected nodes through some way, them is considered as representative of communities.

Social networks can be classified according to two types: *1-mode networks*, which represent the relationship between social entities of the same type, for example, who is friend of who, who depends on whom; and *2-mode networks*, which represents relationships between different social entities, for example, the people who had been to a meeting, the developers that had corrected a given bug. It is important to underline here that from a 2-mode network, it is possible to easily generate a 1-mode network [20].

4 The OSSNetwork Environment

The OSSNetwork environment allows one to: (i) retrieve information from FLOSS repositories, (ii) store this information in a database, (iii) generate different social networks from this information, and, finally, (iv) analyze these networks using tools to manipulate, edit, and execute algorithms. This can be seen in the Figure 2.

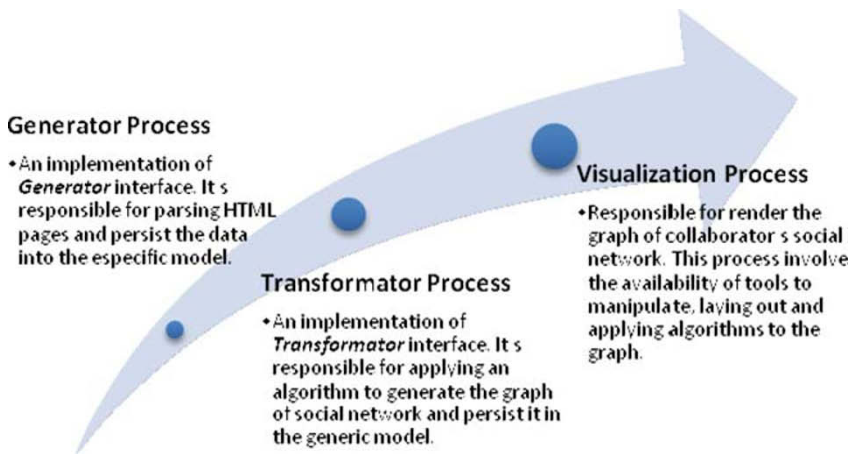


Figure 2 – The OSSNetwork approach

The OSSNetwork is implemented in Java and uses matrices to generate the social networks from the data extracted from projects repositories. Data is extracted through parsing HTML information about forums, mailing lists, bug tracking and chat.

4.1 An Example: the Apache Jackrabbit Project

Figure 3 below illustrates the social network generated with the OSSNetwork environment from the messages exchanged in the mailing list of Apache Jackrabbit

project. In this figure, it is possible to observe some features already implemented, such as: handling of vertices and edges, use of geometric shapes and sizes according to some metrics, annotations, highlight of neighbor vertices, etc.

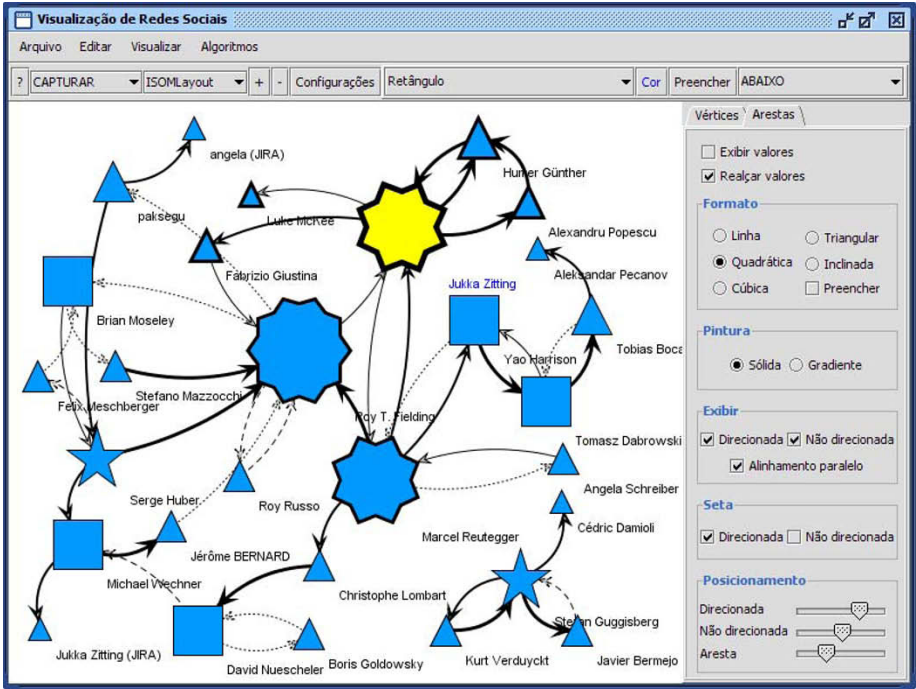


Figure 3 – The Apache Jackrabbit project in the OSSNetwork environment

4.2 Extracting Information from FLOSS Repositories

The OSSNetwork aims to minimize the effort of researchers interested in the social network analysis of FLOSS communities, which requires the extraction of information from these communities. Currently, the OSSNetwork has support for several repositories including: SourceForge, RubyForge Apache, and Microsoft Codeplex.

As mentioned earlier, the environment uses the FLOSSmole framework [14] to parse HTML web pages from the supported repositories. In addition, we implemented new parsing abilities to extract information to be used for social network analysis, for instance, information from the mailing lists which includes who sent a message, who replied to that message, etc. All these steps can be extended providing a new implementation of the *Generator* interface.

4.3 Social Network Generation

Social networks are generated from information extracted from mailing lists, forums and the discussion messages associated with bugs. The mechanism used to generate these networks uses two matrix operations: multiplication and transposition. These operations are required to obtain a 1-mode network from an existing 2-mode network [20]. All generated data related to social networks are stored in a new generic model.

Addition new algorithms to generate different social networks is an easy task. In order to do that, one needs to implement the *Transformator* interface. The *transform* method receives the list of discussion and must execute a calculation over the data and return a graph represented by the generic model of social network that is based on the JUNG framework [21]. Social networks generated by the environment can be exported to files in XML, CSV and DL formats to be used in other social network analysis tools like UCINet [22].

5 Conclusions and Future Work

Empirical research on FLOSS has aroused increasing interest from researchers. Nevertheless, tools that assist in this type of research are still scarce in view of the difficulties inherent in the research process. The OSSNetwork environment described in this paper helps to reduce these difficulties by providing an environment with an integrated set of tools and functionalities to facilitate data extraction, manipulation, and analysis. These functionalities can be extended by new implementations of some interfaces and adding new algorithms to the set of tools currently supported.

We expect that our environment will facilitate the adoption of new approaches using social network analysis. In particular, we are interested in multi-dimensional analysis of social networks extracted from FLOSS communities, that is, analysis where it is possible to take into account, at the same time, different social networks created from different data, such as a public forum, chat rooms, bug tracking systems, mailing lists, etc. We argue that a multidimensional analysis will allow us to study the relationship among different social networks. For instance, the different roles that a same developer has on different aspects (code, bug fixing, user support, etc) of a FLOSS project.

Acknowledgments

This research was supported by the Brazilian Government under grant CNPq 479206/2006-6, by the Universidade Federal do Pará, and by a Microsoft grant.

References

- [1]. **Hoepman, J. and Jacobs, B.** *Increased Security Through Open Source*. Communications of the ACM, pp. 79-83, 2007.
- [2]. **Raymond, E. S.** *The Cathedral and the Bazaar*. s.l. : First Monday, 1998.
- [3]. **Madey, G., Freeh, V. and Tynan, R.** Modeling the F/OSS Community: A Quantitative Investigation. *Free/Open Source Software Development*. s.l. : Idea Publishing, 2004.
- [4]. **Rullani, F.** *Dragging Developers Towards The Core. How The Free/Libre/Open Source Software Community Enhances Developer's Contribution*. Pisa, Italy, 2006.
- [5]. **Jensen, C. and Scacchi, W.** *A Reference Model for Discovering Open Source Software Processes*. Limerick, IR. Third IFIP International Conference on Open Source Systems, 2007.
- [6]. **Halloran, T. and Scherlis, W.** *High Quality and Open Source Software Practices*. Orlando, FL : s.n., Workshop on Open Source Software Engineering, 2002.
- [7]. **Ducheneaut, N.** *The Reproduction of Open Source Software Programming Communities*. Berkeley, CA : UC Berkeley, School of Information Management and Systems, 2002.
- [8]. **Jensen, C. and Scacchi, W.** Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study. *International Conference Software Engineering*. 2007.
- [9]. **Gao, Y. and Madey, G.** *Network Analysis of the SourceForge.net Community*. Limerick, Ireland : s.n., International Conference on Open Source Systems, 2007.
- [10]. **Mockus, A., Fielding, R. and Herbsleb, J.** *A Case Study of Open Source Software Development: The Apache Server*. Limerick, IR, International Conference on Software Engineering, 2000.
- [11]. **Lopez-Fernandez, L., Robles, G. and Gonzalez-Barahona, J.** *Applying Social Network Analysis to the Information in CVS Repositories*. Juan Carlos, Spain : Universidad Rey Juan Carlos, 2004.
- [12]. **Crowston, K. and Howison, J.** *The Social Structure of Open Source Software Development Teams*. Seattle, WA : s.n., International Conference on Information Systems, 2003.
- [13]. **De Souza, C., Froehlich, J. and Dourish, P.** *Seeking the Source: Software Source Code as a Social and Technical Artifact*. Sanibel Island, FL : s.n., ACM Conference on Group Work, 2005.
- [14]. **Howison, J., Conklin, M. and Crowston, K.** FLOSSmole: A Collaborative Repository for FLOSS Research Data and Analyses. *Journal of Information Technology & Web Engineering*. 2006.
- [15]. **Gertz, M.** *Mining Email Social Networks in Postgres*. Shanghai, China : s.n., International Workshop on Mining Software Repositories, 2006.
- [16]. **Conklin, M.** *Do the Rich Get Richer? The Impact of Power Laws on Open Source Development Projects*. Portland, Oregon : s.n., Open Source Conference (OSCON), 2004.
- [17]. **Xu, J., et al.** *A Topological Analysis Of The Open Source Software Development Community*. Big Island, Hawaii : IEEE Computer Society, HICSS, 2005.
- [18]. **Kadushin, C.** *Introduction to Social Network*. 2004.
- [19]. **Hanneman, R. and Riddle, M.** *Introduction to Social Network Methods*. Riverside, CA : University of California, 2005.
- [20]. **Wasserman, S. and Faust, K.** *Social Network Analysis: Methods and Applications*. Cambridge, UK and New York : Cambridge University Press, 1997.

- [21]. **Fisher, D., et al.** Analysis and Visualization of Network Data Using JUNG. *Java Universal Network/Graph Framework*. [Online] http://jung.sourceforge.net/doc/JUNG_journal.pdf.
- [22]. **Borgatti, S., Everett, M. and Freeman, L.** UCINET 6 Social Network Analysis Software. *Analytic Technologies -- Social Network Analysis & Cultural Domain Analysis*. 2006.

Reflection on Knowledge Sharing in F/OSS Projects¹

Sulayman K. Sowe and Ioannis Stamelos
Department of Informatics, Aristotle University, Greece
sksowe,stamelos{@csd.auth.gr}

Abstract. Knowledge sharing between software project participants simplifies a range of decision-making processes and helps improve the way software is being developed, distributed, and supported. However, research in this area has traditionally been very difficult because the source of knowledge, the code, has been a guarded secret and software developers and users inhabit different worlds. F/OSS projects have changed the way we perceive and understand knowledge sharing in distributed software development. This short paper presents our current understanding, and what needs to be done in terms of empirical research in knowledge sharing in F/OSS projects.

Keywords: Open Source Software Development, Open Source Software Projects, Knowledge Sharing, Knowledge Resources, Collaborative Software Development

1 Introduction

Free and Open Source Software (F/OSS) has changed the way we initiate software projects, develop, distribute, and maintain software. In a bazaar styled project roles are not clearly defined and software users are treated as co-developers. The cohabitation of both developers and users facilitates communication between all individuals involved in the software development process, thus, making it easy to study the knowledge sharing activities. Many F/OSS projects succeed because people volunteer their time and effort to share their knowledge and develop the software. Interaction between participants in the form of “talking to each other” is important for cooperation, resolution of conflicts, and the establishment of trust between project participants [1]. If F/OSS is all about coding and few project members are willing to interact and share their knowledge, questions one may ask are;

¹ This research is partly funded by the FLOSScom project (<http://flosscom.net/>). Grant No. 229405-CP-1-2006-1-PT-MINERVA-MPP

Please use the following format when citing this chapter:

Sowe, S.K. and Stamelos, I., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 351–358.

- effective communication and knowledge creation are used as measures of project success [2,3]. But is there knowledge sharing, who are the people involved, how much knowledge sharing are developers and users doing?
- how can participants at the very edge of the project's community feel that they are part of the project when they are not opportune to interact and share their knowledge with the rest?

The first step in answering these questions is to understand the nature and dynamics of knowledge sharing in F/OSS projects. The rest of the paper attempts to develop such understanding. First we argue for a paradigm shift from traditional knowledge sharing practices. Then we address knowledge sharing in F/OSS projects and use some empirical studies to reflect on our current understand and map future research directions.

2 Paradigm Shift in Knowledge Sharing in F/OSS

The importance of knowledge sharing is not unique to F/OSS projects alone. Software projects face a problem of how to leverage and transform the tacit knowledge of community members into explicit usable knowledge. Practically, a successful project needs to manage its knowledge resources (people and technologies) in order to generate an efficient and sustainable software development environment. Knowledge management (KM) practices often consider how organizational structures and processes promote collaborative learning [6] and sharing of knowledge [4]. The goal of knowledge management is to lower barriers imposed by organizational structures, format restrictions, and conceptual limitations, so that the acquisition, transfer, and sharing of knowledge can be facilitated. Thus, KM helps us understand tools, roles, and activities we need to develop in order to help individuals benefit from F/OSS. However, F/OSS projects are different entities when compared to organizational structures in which many KM studies are conduct. To address KM issues in F/OSS projects, one needs a *paradigm shift* from organizational attention and knowledge making as a manufacturing process to online interactive systems where knowledge making is characterized by the virtual presence of geographically distributed groups of volunteers. The virtual world makes hording and selling knowledge difficult, if not impossible, because someone else might be positioned to provide the same knowledge for free.

The F/OSS development process not only highlights a different way of sharing knowledge but also acquiring it is a complicated process and building on existing knowledge takes time. Software development is a human-based and knowledge-intensive activity [8,9]. In addition to sound technological infrastructure and effective tools to coordinate the collaborative software development process [1], the success of any project immensely depends on the knowledge and experience brought to it by software developers and users. F/OSS projects provide the means for members to transform their tacit knowledge into explicit [4] (**Fig. 1**). As illustrated, project participants socialize by sharing their knowledge. Individuals make their tacit

knowledge explicit to the project through externalization. Combination refers to the formation and organization of abstract knowledge from explicit knowledge. Through internalization individuals will absorb explicit knowledge, combining it with their own knowledge and experiences to produce “new” tacit knowledge. Through active participation, individuals’ tacit knowledge is transformed into explicit knowledge.

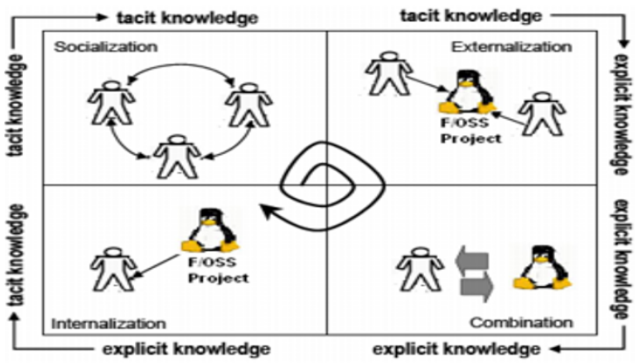


Fig. 1. Making tacit knowledge explicit in F/OSS projects

F/OSS projects are complex environments with many different actors. There is no indication that all project participants will interact and share their knowledge. When members do, *knowledge link (K)* is said to exist between them. The link is a reciprocal relationship of the form *A talks to B* and *B talks to A* and can be expressed as; $K_{AB} = \begin{cases} 1 \\ 0 \end{cases}$; where $K_{AB}=1$ if there is knowledge sharing between actor *A* and actor *B*, and 0 if otherwise.

Sharing knowledge is a synergistic process. If a project participant shares his ideas or a way of configuring particular software with another person, then just the act of putting his idea into words will help him shape and improve his idea. If he enters into a dialogue with the community, then he may benefit from their knowledge, from their unique way of doing things and improve his ideas further. The benefit derived from knowledge sharing is that participants learn from each other, and the result of their interaction is archived in the project’s repositories from which subsequent participants can learn.

3 Knowledge Sharing in F/OSS Projects

The F/OSS development process facilitates the creation, diffusion, and transformation of software knowledge at a rate unprecedented in the history of software development. The prospects for expert software developers and novice users to understand the software development process are now great. However, our understanding of knowledge sharing in F/OSS projects is challenged by the fact that individuals or even infrastructures are often located at large distances from each other, making tra-

ditional face-to-face knowledge sharing practices hard. Furthermore, the “babbling bazaar of differing agendas and approaches” [10] in some projects has people with varying knowledge and skills. Nevertheless, the F/OSS development process has made the empirical study of knowledge sharing possible because we can now use the freely available data in mailing lists, forums, code versioning systems, bug trackers, etc. to understand who is involved, who is talking to whom, and what is talked about. The F/OSS development process is knowledge intensive and involves real people who create, maintain and evolve the software. Therefore, focusing on knowledge sharing in F/OSS projects may shed light on pending research issues such as motivation and involvement of developers and users, community and software project formation and dynamics, reasons for projects success or failure, software engineering practicing, learning opportunities, how participants’ contributions change overtime, etc.

For many individuals, developing new skills and sharing knowledge are primary motives for participating in F/OSS. Through reflective practice and collaborative learning, participants strive to improve the software in their respective projects, develop and refine their skills, challenge and question themselves. There are many opportunities for serendipity or information encountering in F/OSS projects. For example, a potential knowledge seeker [6] confronts an unfamiliar concept (or bug) in the use of an application (e.g. OpenOffice) and decides to seek help form a project’s mailing list. There are two ways to look at this scenario. First, if the concept has been encountered before, it will be captured and stored in the project’s knowledge base (mailing lists, forums, code repository, etc). He/she then consults and learns from the knowledge base. Second, if the concept has not been encountered, ie. the knowledge seeker’s problem has not been addressed in the project before. He/she then identifies the appropriate list, posts his/her question, and exchanges ideas with list participants. Alternatively, the knowledge seeker may know a project participant with expertise in the area he is interested in and exchange direct emails with that person, with or without copies being posted to the list.

4 An Empirical Investigation of Knowledge Sharing

Methodologies suitable for investigating knowledge sharing in terms of email exchanges have been proposed [5, 6]. The intangible nature of tacit knowledge makes it very difficult, if not impossible, to measure. However, when tacit knowledge is transformed into explicit knowledge through socialization or interaction and shared by members of a project, an attempt can be made to measure how much knowledge is being shared. One approach to quantify the level of knowledge sharing in F/OSS is by analyzing substantial email exchanges between list participants. This can be achieved by

- (i) Counting the total number of posts externalized to the list. That is, email messages potential knowledge seekers posted. We represented this value by the *nposts* variable.

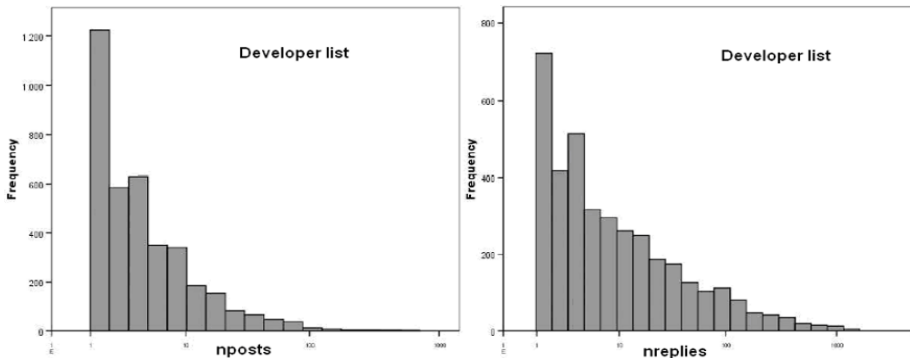
- (ii) Counting the total number of replies made by potential knowledge providers to questions posted to the lists. This value is represented by the *nreplies* variable.

The data (**Table 1**) collection, extraction, and cleaning methodology for investigating knowledge sharing in mailing lists is obtained from the Developer and User mailing lists of the Debian project [6]. The dataset was collected over 5 years (January 2000-December 2005) and consists of 3735 participants in the Developer list, who posted 29685 email messages, which generated 128933 replies. The 5970 participants in the User list posted 193276 email messages, which generated 165380 replies.

Table 1. Descriptive statistics of Developer and User mailing lists data.

		Developer	User
<i>nposts</i>	Mean	7.95	32.37
	Median	3.00	7.00
	Std. Dev.	21.302	121.753
	Maximum	523	4106
<i>nreplies</i>	Mean	34.52	27.70
	Median	6.00	5.00
	Std. Dev.	105.567	122.040
	Maximum	1517	4168

Comparatively, the mean (posts/person) and median of *nposts* are smaller for the Developer list, while the same measures have larger values for *nreplies*. Thus, the posting and replying activities of the participants are different in the two lists - further clarified in **Fig. 2**.



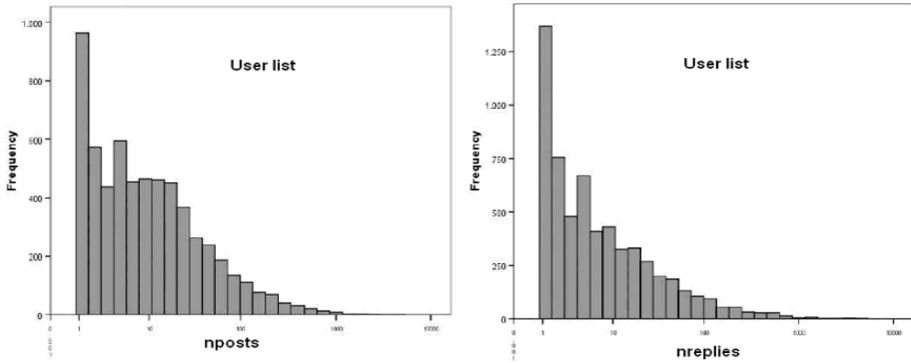


Fig. 2. Frequency distributions of posts and replies

Participants in the User list contributed small posts and small replies. For the developer list, participants’ activities were characterized by a small number of posts and a large number of replies. For the relationship between posting and replying activities, Table 2 shows only the Kendall’s *taub* and Spearman’s *rho*, since we ranked the participants according to their posting or replying [7].

Table 2. Nonparametric correlations between posts and replies in the two lists (Sig. =0.000 for all correlations).

			Developer List		User List	
Test	Variable		nposts	nreplies	nposts	nreplies
Kandella’s <i>taub</i>	nposts	Corr. Coef.	1,000	,475*	1,000	,550*
	nreplies	Corr. Coef.	,475*	1,000	,550*	1,000
Spearman’s <i>rho</i>	nposts	Corr. Coef.	1,000	,608*	1,000	,699*
	nreplies	Corr. Coef.	,608*	1,000	,699*	1,000

5 Reflection on Knowledge sharing in F/OSS Projects

Research findings on developer and user activities in F/OSS projects mailing lists point out that a small number of individuals are responsible for most of the work. This inequality in contribution raises serious concerns that concentration of development effort on a few will limit knowledge sharing in F/OSS projects [5]. The paradox is that this kind of inequality was observed in many successful projects (eg. Apache, KDE, Friefox).

- **Are Developer and User mailing lists participants doing more posting than replying to questions posited to their lists?** In the Developer list participants contributed more replies (mean=34.52) than posts (7.95). The reverse was the case in the User list. Participants posted more than they replied to questions asked in the list. One explanation for this could be that postings in the developer mailing list may contain sufficient information to allow other participants to analyze the request and are given the highest priority since the

reporter is likely to be a member of the development community. Such posts receive the attention of almost all participants.

- **Is there a trend in the way individuals post and/or reply to questions in Developer and User lists?** About 32.8% (1224) of the participants in the Developer list posted one email message and 19.3% (721) contributed one reply. For the User list 16.1% (963) of the participants posted one email message and 22.9% (1369) contributed one reply. The maximum email messages posted (*nposts*) by one individual was 523 in the Developer list and 4106 in the User list. For the replies, the maximum values were 1517 in the Developer list and 4168 in the User list. In none of the lists did the individual who posted the most emails also made the most replies. For example, in the Developer list, the participant who posted the most emails (523) contributed 574 replies. While the participant with most replies (1517) contributed 79 posts.
- **How are the posts and replies of Developer and User mailing lists participants correlated?** In both lists we found that posting and replying activities of the lists participants are correlated. When posting increases, replying likewise increases. However, the correlation was stronger for the User list ($\rho=0.699$). This means that, for the User list more questions generate more answers.

6 Summary and Conclusion

A reflection on the empirical investigation revealed inequality in posting and replying activities in the lists studied. Investigating who was asking versus who was replying, we found out that, in the Developer list, the participant who posted the most emails (523) contributed 574 replies. While the participant with most replies (1517) contributed 79 posts. A similar trend was observed in Apache, Mozilla, KDE. What is different in this case is that we looked at knowledge sharing from the perspectives of both developers and users. Yet still, few individuals are involved in the knowledge sharing process. However, while this may be true for one kind of repository (mailing lists), the situation might be different in another kind, bug trackers or version control systems, for instance. An interesting research direction may be to simultaneously study developer and user lists in big and small, successful and unsuccessful projects to see if the pattern of knowledge sharing reported can be generalized. In addition the role of knowledge sharing for open source project success or failure should be better understood and assessed.

References

1. S. K. Sowe, I. Stamelos, I. Samoladas (Eds.) (2007). *Emerging Free and Open Source Software Practices*. Idea Global, May 2007, pp: vi-vii, 98-119.
2. Karl Fogel. (2005). *Producing Open Source Software: How to Run a Successful Free Software Project*, Creative Commons.

3. Crowston, K., Hala, A., and Howison, J. (2003). Defining Open Source Software Project Success. *Proc. of International Conference on Information Systems, ICIS*.
4. Nonaka, I. & Takeuchi, H. (1995). *The Knowledge Creating Company*. Oxford University Press.
5. Kuk, G. (2006). Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List, *MANAGEMENT SCIENCE*, Vol. 52, 1031-1042.
6. Sulayman K. Sowe, I. Stamelos, L. (2008). Angelis. Understanding Knowledge Sharing Activities in Free/Open Source Software Projects: An Empirical Study, *Journal of Systems and Software*, Vol. 81(4), 431-446.
7. Sheskin, D. J. (2000). *Handbook of parametric and nonparametric statistical procedures*, Chapman & Hall, 491-508.
8. S. K. Sowe. An Empirical Study of Knowledge Sharing in Free and Open Source Software Projects. *PhD thesis*, Department of Informatics, Aristotle University, Greece, 2007.
9. A. Aurum, R. Jeffery, C. Wohlin, and M. Handzic, editors. *Managing Software Engineering Knowledge*. Springer, 2003.
10. S. E. Raymond. *The Cathedral and the Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, Sebastopol.

Usability in Company Open Source Software Context - Initial Findings from an Empirical Case Study

Netta Iivari¹, Henrik Hedberg¹, and Tanja Kirves¹

¹ University of Oulu, Department of Information Processing Science
P.O.Box 3000, FIN-90014 University of Oulu, Finland
{netta.iivari, henrik.hedberg}@oulu.fi, tanjakir@mail.student.oulu.fi

Abstract. In the company open source software (OSS) development context, usability is becoming an important issue due to a growing user population, who is only interested in usable applications, not in their development. Companies try to gain a competitive advantage of OSS by utilizing available components, but the openness is difficult to achieve in the business world with licenses, patents and intellectual property rights. This paper analyses usability and user-interface (UI) development in the company OSS context through an interpretive case study in a software development unit of a large, global corporation. Our initial findings suggest that there is a need for human computer interaction specialists in the OSS context. It is revealed that with the software based on OSS, more time can be spent on usability and UI design. In the company's viewpoint, there are still many issues involved in deciding what parts of the product will be open source. Especially UI code may remain closed due to competitive advantage and patents.

1 Introduction

This paper analyzes how usability and user-interface (UI) development are dealt with in company open source software (OSS) development context. Company OSS development context refers to 'OSS 2.0', i.e. to the new, commercially viable OSS development, in which software (SW) development companies try to gain a competitive advantage of OSS [7]. SW development companies release the source code of their products and produce OS compliant licenses, and OSS companies develop their business around raising money through licensing [7], [15]. In OSS development, the source code needs to be 'available for anyone who wants to use or modify it', but in the company OSS context, there actually is a 'continuum of openness' [16: 151]. This paper focuses on commercial SW development utilizing OSS as part of the product, and potentially also releasing the source code for the OSS community to develop it further.

Nowadays OSS is targeted at a mass of users, not only for developers developing it for their own use. The user population of OSS is becoming larger, including a growing number of non-technical users, who are not interested in OSS development, but only in the resulting solution [4], [8], [14], [15], [18]. Therefore, usability and

Please use the following format when citing this chapter:

Iivari, N., Hedberg, H. and Kirves, T., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 359–365.

high quality UI are becoming an important issues also in OSS context [1], [14], [19], [20]. This paper reports initial findings of an interpretive case study designed to analyze usability and UI development in company OSS development context.

2 Usability and User Interface Development

The importance of usability and UI development are emphasized especially within the field of human computer interaction (HCI). Usability refers to 'the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use' [10]. Another widely cited definition defines usability to consist of learnability, efficiency, memorability, errors and satisfaction; it should be easy to learn to use the system, its use should be efficient, it should be easy to remember how to use it, the users should not make errors or they should easily recover from them, and the use should be satisfactory. [16]

Different user-centered design (UCD) and usability engineering (UE) methodologies offer guidance for usability and UI development. Many differences can be identified in them, but they all emphasize the importance of gaining a detailed understanding the user, his/her tasks or work practices and the context of use as a basis. Afterwards, one should carefully redesign the tasks or work practices based on that understanding. User involvement in the design process is argued for. User feedback should be gathered early, and the design solution should be iterated based on the user feedback. [2], [12], [16], [17]

To carry out these activities, the HCI literature suggests hiring a group of specialists, variably labeled e.g. as usability, UCD or UE specialists. They are to carry out or facilitate the analysis of the users, their current tasks and work practices and the context of use, the user task redesign and the iterative usability evaluations. They are also to ensure active user involvement. [2], [9], [12], [16], [17]

Many challenges involved with usability and UI development in the OSS development context have already been revealed. In OSS development, the developers typically produce the SW for themselves. From the viewpoint of non technical users, usability of the OSS tends to be poor, and the development process anything but 'user centered' [1], [3], [4], [14], [19], [20]. Therefore, usability is becoming a relevant topic of research in the OSS context, even though, to date, it has not been examined much [1], [4], [14], [19].

Typically, the OSS developers do not have knowledge about the non technical users, their tasks and the context of use. In addition, no UCD/UE methodology is typically employed. Extensive gathering of user feedback and bug reporting have been highlighted as solutions in OSS context. However, it has also been reported that non-technical users may be intimidated or incapable to report bugs. [1], [3], [4], [13], [14], [19], [20] HCI specialists do not typically participate in the OSS development, and the OSS developers do not normally have the HCI knowledge and skills needed. The users can not replace HCI specialists either, because they are not trained for

developing and ensuring usability, even though they encounter the usability problems while using the system. [1], [4], [8], [13], [14], [19] [20].

Related to company OSS development, it has already been argued that large corporations that participate in OSS development can provide professional usability resources and HCI guidelines for OSS development [1]. However, there clearly is a lack of research on company OSS development, and particularly on usability and UI development in this context. This paper provides insights into this phenomenon.

3 Empirical Case

In this research effort an interpretive case study method is utilized. Case studies are suggested, if little is known about the phenomenon or current perspectives have little empirical evidence or conflict with each other [6]. There is a lack of research on usability in company OSS development context. Interpretive case study method was selected, since through it one can focus on the meanings attached to the phenomenon studied from 'the native's point of view', produce thick descriptions, and gain thorough understandings. The focus is on understanding and making sense, not explaining in the predictive sense. Theories are used as sensitizing devices. [5], [11]

The case involved in this study is a SW development unit of a large, global corporation that has been involved with OSS development already for few years. It has a strong background in usability and UI development. However, now the company is facing challenges in combining the worlds of commercial SW development, OSS development and usability and UI work. The case was selected, since it offers a rich setting to analyze the emerging 'company OSS' phenomenon.

Different kinds of empirical data have been gathered from the unit. This paper relies on data gathered by interviewing the personnel and by keeping field notes after joint meetings organized in the unit. Five persons were interviewed, including a project manager (denoted as [a]), two developers ([b] and [c]) and two HCI specialists ([e] and [f]). In addition, field notes from three meetings organized with the management of the unit were included in the analysis. This paper reports our initial findings related to how usability and UI development were dealt with in this particular case of company OSS.

The OSS was seen as a basis to build up something new in the company: *"From the viewpoint of the firm it is fast, time is not spent in implementation, but one gets everything ready made. One can concentrate on doing new things, not things that already exist."* [c] Typically, available solutions were evaluated and an OSS alternative chosen that was further developed to fit the company's needs. That way the development time was reduced significantly: *"Without this communication and library, [an application] would not have been possible. There was 6 months time, and it took 3 months, because there was support from the community. Otherwise it would have taken over a year."* [c] In addition to the support for the implementation of the low-level library, the OSS community gave feedback that was important for

usability of the product: *"From the viewpoint of UI design there are more users, a social network of knowledge."* [e] Interestingly, OSS was seen as a source of knowledge: *"[OSS is] learning and collaboration, a place to learn from developers."* [e] *"One always learns something. One sees other people's code, their expertise."* [c]

Usability and UI design were emphasized in the case: *"[Usability is] while developing UI, one of the most essential things"* [a]. *"[The firm] makes devices for people. Why would anybody buy a product that was difficult to use, compared to competitors? So market is the reason [for usability]"* [b]. *"[Usability is] extremely important: competitive edge, brand, image."* [d] There have been recent changes to emphasize the importance of usability and UI design: *"Usability and user experience are the key, the company started to invest in them. (...) In the beginning, one person took care of everything. Now there are seven persons in the team, so the role is bigger. (...) Power has been given to designers, from developers. They have the power to figure out things, they are very concerned with usability."* [e]

The difficulties involved with usability and UI design in OSS development are acknowledged: *"Developers produced the UI. It was difficult to help them, if you were not there in the beginning. (...) [I] tried to change the code in an OSS project, but it can be bureaucratic in there, they get a lot of change requests, they do not necessarily want to implement them."* [e] *"Usability is hard in OSS projects. There are no usability professionals. (...) The main problem is the usability staff in OSS projects. Except when big companies support the OSS projects. Because the firms are interested in the end user and give help."* [c]

Therefore, the support gained from companies to OSS projects is highlighted. Especially the importance of professional HCI specialists' participation in OSS projects is emphasized: *"[Interaction designers] there should be more of them willing to help, UI designers to plan and design, and developers to sponsor them."* [e] *"Get a UI designer and a usability specialist"* [b]. There should be *"open usability work: this kind of persons could be there commenting and helping (...) Usability specialists could gain reputation, if it results in good products"* [d] *"One should attract also designers, artists, end users. All areas. Developers learn from these [other areas]"* [c].

The usage of the OSS gave more time for the usability and UI design: *"One could much earlier experiment with the finished software. One did not just use own simulation. This was a positive effect."* [d] *"Late changes are possible, because own process resembles OSS projects. OSS enables late changes. There is more design and evaluation. So, it helps usability that there is code ready-made. Time is not spent on that, you start building on top of that."* [e] *"[OSS development is] faster and closer to user"* [c]. In addition, OSS gave access to users: *"In OSS development one is always in contact with the user. One knows what they want and one knows if the wishes change."* [c] *"OSS community, users -> a rich place to gather user feedback. User-centered design in the community: wishes are ranked, [the users] are used as partners in the beginning and also during the design"* [d]. Being in contact with user is *"easier and cheaper"* [e].

UI design was a key to success: *"Designers (...) always thought of users, they thought of interaction, so that the users don't get tired. We have succeeded well. (...) Best thing of the whole project is the UI."* [c] Actually, the UI part was seen as a competitive edge for the company, and that was one factor affecting not to publish the source code: *"Usability and UI are so valuable that they are not released."* [e] The chosen model was to share improvements made into the OSS library and to keep the UI part of the application closed: *"[An application is] closed source, on top of open source."* [e] That fulfilled the license terms of the original source code also: *"Openness did not spread to the UI code."* [a] Despite the situation, the company continued to interact with the community: *"There is cooperation with the community, gathering of feedback, even if it was closed source. Bugs are fixed in the OSS."* [e]

The decision had negative impacts in the community: *"If the UI is closed that causes anger in some people"* [e], but the company had also other reasons for their decision: *"People asked for the source code, but it was not given. It was an IPR risk."* [a] Everything had to be analyzed keeping the company secrets in mind: *"In the beginning, sourceforge's cvs server was used, and everything was not thought of. It was not thought of that what is secret, proprietary (...) It was moved to the firms own server, the in-house made component was put in there also. Legal check must be made. This component is not in sourceforge."* [b] The most secure way to act is to develop the product in-house first, and then release it, if it is possible: *"Bug fixes are given back, but if there are risks, a legal check is done. Everything is not let out."* [e]

In general, the licenses were seen as an issue in company OSS development: *"License defines the openness. It is difficult"* [a] *"Sometimes there are difficulties due to the licenses. One has to know what kind of code is used, what libraries are used. Usually firms know these issues, they have to know. (...) There needs to be lawyers. One has to hire them."* [c] Sometimes the procedures needed, before the open source release can be made, were seen as too expensive and complicated, when compared to the benefits: *"[The company] has a lot of patents related to [issues related to the OSS and the application], there was no desire to carry out investigations."* [a] That encourages to keep the SW as closed as possible. The result is a hybrid SW with open and closed source parts.

4 Concluding Discussion

In the company OSS context, OSS solutions are used as part of the products, but there are many issues involved in deciding what parts of the product will be open source. Especially UI code tends to remain closed. Several reasons were identified related to that. Difficulties with licenses and patents were brought up. However, interestingly, UI was also seen as a competitive advantage, which was for that reason kept closed source. It was maintained that by putting effort into the UI design, the company can beat the competitors. However, for that reason, it is unlikely that the UI parts become open source in the company OSS context.

Regarding the field of HCI, the results of this study seem promising. If usability and UI design are becoming such central concerns in SW development as indicated by this study, the field of HCI will increase its importance. There probably will be a need for HCI methods and tools, which are particularly tailored to fit the OSS development context, as well as for HCI specialists contributing to the development. Regarding the relationship between HCI and OSS, two interesting observations are emphasized. First, the results confirm that there is a need of HCI specialists in the OSS context. The interviewees emphasized the need of professional HCI specialists (some specializing in usability, others in UI design), taking care of understanding the users, their tasks and the context use, and evaluating the solution with the users.

Another interesting issue was the benefits of OSS that can be gained by the HCI specialists in the company OSS context. It was emphasized that realistic user feedback can be gathered earlier than normally if part of the product is OSS. Therefore, OSS may reduce the development time, which enables early usability evaluation with the actual SW. In addition, it was emphasized that due to the reduced development time, more time can be spent on usability and UI design, which is also a clear benefit from the HCI viewpoint. Moreover, the possibility to concentrate on innovating new things, when utilizing OSS components as a basis, was brought up as a benefit of OSS in the company context.

This paper provided initial findings from our empirical examination. More thorough analyses will be carried out in the future. Paths for future work include also deepening the understanding of usability in the company OSS context, and devising procedures that enable HCI specialists to participate in OSS communities. In general, the effects of usability work on OSS products, communities and companies, and the means to achieve those effects are interesting, momentous research topics.

References

- [1] Benson C., Müller-Prove, M., & Mzourek, J.(2004): Professional usability in open source projects: GNOME, OpenOffice.org, NetBeans. Extended Abstracts of the Conference on Human Factors in Computer Systems. New York, ACM Press. Pp. 1083-1084.
- [2] Beyer, H. & Holtzblatt, K. (1998). Contextual Design: Defining Customer-Centered Systems. San Francisco: Morgan Kaufmann Publishers.
- [3] Bødker, M., Nielsen, L. & Orngreen, R. (2007): Enabling User-Centered Design Processes in Open Source Communities. In N. Aykin (ed.): Proc. Human Computer Interaction International, Part I: Usability and Internationalization. LNCS 4559. Pp. 10-18.
- [4] Cetin, G., Verzulli, D. & Frings, S. (2007): An Analysis of Involvement of HCI Experts in Distributed Software Development: Practical Issues. In D. Schuler (ed.): Proc. Human Computer Interaction International: Online Communities and Social Computing. LNCS 4564. Pp. 32-40.

[5] Denzin, N. & Lincoln, Y. (2000): Introduction: The Discipline and Practice of Qualitative Research. In N. Denzin, Y. Lincoln (Eds.): Handbook of Qualitative Research. 2nd edition. Thousand Oaks: Sage Publications.

[6] Eisenhardt, K. (1989): Building Theories from Case Study Research. *Academy of Management Review* 14(4): 532-550.

[7] Fitzgerald, B. (2006): The Transformation of Open Source Software. *MIS Quarterly* 30(3), 587-598.

[8] Frishberg, N., Dirks, A., Benson, C., Nickel, S. & Smith, S. (2002): Getting to know you: open source development meets usability. Extended Abstracts of the Conference on Human Factors in Computer Systems. New York: ACM Press. Pp. 932-933.

[9] Iivari, N. (2006): Discourses on 'culture' and 'usability work' in software product development. *Acta Universitatis Ouluensis, Series A, Scientiae rerum naturalium nro 457*.

[10] ISO 9241-11 (1998): Ergonomic requirements for office work with visual display terminals (VDT)s - Part 11 Guidance on usability. International standard.

[11] Klein, H. & Myers, M. (1999): A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly* 23(1), 67-94.

[12] Mayhew, D. (1999): The Usability Engineering Lifecycle: A practitioner's handbook for user interface design, San Francisco: Morgan Kaufmann Publishers.

[13] Nichols, D., Thomson, K. & Yeates, S. (2001): Usability and Open Source Software Development. Proc. Symposium on Computer Human Interaction, Palmerston North, New Zealand, July 6. Pp. 49-54.

[14] Nichols, D. & Twidale, M. (2006): Usability Processes in Open Source Projects. *Software Process Improvement and Practice* 11, 149-162.

[15] Niederman, F., Davis, A. Greiner, M., Wynn, D. & York, P. (2006): A Research Agenda for Studying Open Source I: A Multilevel Framework. *Communication of the Association for Information Systems* 18, 129-149.

[16] Nielsen, J. (1993): Usability Engineering, Boston: Academic Press.

[17] Rosson, M. & Carroll, J. (2002): Usability Engineering: Scenario-based Development of Human-Computer Interaction, San Francisco: Morgan-Kaufman.

[18] Viorres, N., Xenofon, P., Stavrakis, M., Vlanhogiannis, E., Koutsabasis, P. & Darzentas J. (2007): Major HCI Challenges for Open Source Software Adoption and Development. In D. Schuler (Ed.): Proc. Human Computer Interaction International: Online Communities and Social Computing. LNCS 4564. Pp. 455-464.

[19] Zhao, L. & Deek, F. (2005): Improving Open Source Software Usability. Proc. 11th Americas Conference on Information Systems, Omaha, USA, August 11-14. Pp. 923-928.

[20] Zhao, L. & Deek, F. (2006): Exploratory inspection: a learning model for improving open source software usability. Extended Abstracts of the Conference on Human Factors in Computer Systems. New York: ACM Press. Pp. 1589-1594.

Willingness to Cooperate Within the Open Source Software Domain

Pascal Ravesteyn and Gilbert Silvius¹
University of Applied Sciences Utrecht, The Netherlands
pascal.ravesteijn@hu.nl, gilbert.silvius@hu.nl

Abstract. Open Source Software (OSS) is an increasingly hot topic in the business domain. One of the key benefits mentioned is the unlimited access to the source code, which enables large communities to continuously improve a software application and prevents vendor lock-in. How attractive these benefits may be, the market for OSS however remains limited. In the Netherlands research consisting of 7 interviews and a survey among 206 Open Source Software Service providers (with a 34% response rate) was done to determine whether service providers wanted to cooperate in an Association that will set quality levels and guarantees to its members and their customers.

Keywords: Open Source Software, Communities, Quality

1 Introduction

In the last decade open source software (OSS) initiatives have been steadily growing, resulting in more and more companies that provide services, support and certification around open source applications. From a user perspective some of the most important reasons for the use of OSS are: cost effectiveness, improved flexibility, expiration of maintenance, availability of support through both software vendors and service providers, independence from software vendors, increased technical requirements, increased interoperability, security aspects and improved reliability (Ghosh et al. (2005). Probably the most important of these characteristics is the independence from software vendors which means there is no or limited vendor lock-in (Pavlicek, 2000; Raymond, 2001; Wichmann, 2002b; Goldman et al., 2005). Vendor lock-in implies that it is very hard to switch to other software and/or service providers due to high switching costs or the usage of legacy and non-standard software that is available only through the network of one vendor. Because open source software is normally based on open standards and open interfaces (Varian et al., 2003) it becomes easier to migrate to different software products. Normally communities evolve around open source software that then adapt and further develop the applica-

Please use the following format when citing this chapter:

Ravesteyn, P. and Silvius, G., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 367–373.

tions and services. Although users of open source software are not dependent on a single vendor or service provider to deliver updates with new functionality, in practice an emotional binding with one supplier still seems to exist.

Contrary to the reasons that are found for end-user adaptation, the trigger for many software providers to open sourcing their offerings are mostly internally motivated (Wijnen-Meijer and Batenburg, 2007) and based on market position, the capability for product innovation (or lack thereof) and the degree of customer independence. This seems to be supported by Cusumano (2004), who in his book *The Business of Software* describes several product characteristics that may be relevant for the decision of open sourcing such as (1) the fundamental difference between intended audiences: enterprises and home users (2) the difference between niche and mass audiences, and software with a horizontal or a vertical functional scope (3) the market position of a software product. This can be leading, complementary or following. Also some software providers start with using open source software internally because of its perceived cost effectiveness (Grand et al., 2004) before considering open sourcing their own offerings. Finally governments are an important trigger to vendors to provide open source software due to the value they give open source software for its conformance to open standards that help to ensure accessibility of governmental information (Varian et al., 2003).

To bridge the gap between the motivation given by end-users versus software vendors and service providers on why to adopt the open source paradigm, Rijke (2005) suggests that open source software vendors and service providers should cooperate in a more structured way to provide improved flexibility, maintenance, availability of support, increased technical requirements, increased interoperability, improved reliability and higher quality to end-users. This suggestion is based on the fact that a large majority of the open source vendors and service providers in the Netherlands are small office and home office companies. Contrary to the numerous studies on the reasons for OSS (Ghosh et al., 2005; Wichmann, 2002a), very little research is available on the cooperation between organizations and what the triggers for such cooperation are in the OSS domain. This research tries to find an answer to the question if organizations within the open source domain are willing to cooperate with their peers to improve the different aspects as mentioned with a focus on the improvement of maintenance, support, reliability and quality.

The following section elaborates further on the market for open source products and services in the Netherlands. Then the research methodology that was used is described and the results from both the interviews and the survey are given. In the final section the limitations of this research will be mentioned and some suggestions for future research are given.

2 Open Source Software Market in the Netherlands

At the start of this research only part of the Dutch market of open source software and service providers was known. This meant that the first activity was to exten-

sively map the entire market. This was done by looking at different available resources such as the governmental Program for Open Standards and Open Source Software, lists of delegates to different Open Source conferences that were available and the Internet. Besides creating an overview of the organizations that are active in the Dutch market we also determined the involvement of these organizations within the open source communities that already existed. The rate of involvement is measured on a 3 point scale (low, average and high score) and was determined based on the number of times delegates of these organizations attended conferences (e.g. Holland Open), were active speakers or were involved in professional publications. The final result of this part of the research was an extensive overview of 222 organizations that are active in the open source domain in the Netherlands during 2006. Of each organization information was collected. Figure 1 shows a small part of the matrix that was the end result.

Overview Open Source Software service providers & applications											138	8	2	7	66	4	78	1	3	110	79	79
Company name	Company type	# employees	Contact person	Type of services	# projects OSS	Consultant	Design	Development	Implementation	Testing	Maintenance	Hosting	Other	Education	Other							
ICT Services BV	BV	31-100	2001	consulting, design, development, implementation, testing, maintenance, integration, hosting, education, integration, design, development, implementation, testing, maintenance, integration, hosting, education, integration, design, development, implementation, testing, maintenance, integration, hosting, education	2	2	2	2	2	2	2	2	2	2	2							
Info.nl	BV	6-10	1995	consulting, design, development, implementation, testing, maintenance, integration, hosting, education, integration, design, development, implementation, testing, maintenance, integration, hosting, education	2	2	2	2	2	2	2	2	2	2	2							
Lectra	BV	6-10	2001	education	2	2	2	2	2	2	2	2	2	2	2							
Lectra OSS - the partner for business		2-5	2002	education, implementation	2	2	2	2	2	2	2	2	2	2	2							
Lectra BV	BV	6-10	1997	consulting, design, development, implementation, testing, maintenance, integration, hosting, education, integration, design, development, implementation, testing, maintenance, integration, hosting, education	2	2	2	2	2	2	2	2	2	2	2							
Lectra Software BV	BV	31-100	1988	consulting, design, development, implementation, testing, maintenance, integration, hosting, education, integration, design, development, implementation, testing, maintenance, integration, hosting, education	2	2	2	2	2	2	2	2	2	2	2							
Lectra & Area B.V.	BV	6-10	1999	consulting, design, development, implementation, testing, education	2	2	2	2	2	2	2	2	2	2	2							

Fig. 1. Example of data collected on open source organizations in the Netherlands

3 Research Methodology

Based on the open source market overview it was determined to first do several interviews followed by a survey because by gathering data from different angles a clearer picture of the real world can be modeled and validated (Baarda et al., 2001). The interviews were semi-structured and resulted in a validated list of research questions to be used in the survey.

The interviews had two goals, first they provided us with a validation on the market overview and secondly the outcomes were used to validate and broaden the list of research topics that made up the survey. Seven organizations were sent a first draft of the survey questions to fill out and return, after which an interview was held. The interview was based on the answers the respondents provided before hand and lasted approximately for 2 hours.

The final survey consisted out of 23 questions, some of which had several sub questions. Both open and closed questions were used. Where respondents had to answer on a scale, we used a 4 point scale ranging from completely agree to completely disagree. The survey was sent to 206 mail addresses out of the 222 organizations available in the market overview (some e-mail addresses were missing), and after a re-

minder (two weeks after the first participation request a total of 70 surveys were returned. The final response rate was around 34%.

4. Results

The question that this research tries to answer is: *Are organizations within the open source domain willing to cooperate with each other to improve (amongst others) the maintenance, support, reliability and quality of open source services and software?*

Based on the interviews this is not the case. According to our findings organizations are only willing to cooperate with others when this results in direct financial gains for their own company. Only two respondents also mentioned higher quality of their service or software offering important enough to consider cooperation. But the interviews were mainly used to validate the survey questions and the outcomes can not be considered as the general opinion for the entire sector. The survey however shows a more diverse outcome in the reasons to cooperate.

It is important to know what organizations within the open source domain find to be the strong and weak points of open source software. Therefore we asked what the respondents thought the characteristics of open source software versus closed source software are. Not the strongest points were: no vendor lock-in (89%), everybody is able to improve the software (82%) which results in a higher rate of innovation (84%), and no licensing costs (78.5%). Asked if there were any weak points in open source software there was a large difference in answers with 44% of the respondents stating that the quality of open source is lower than closed source software (with a small majority of 56% who think otherwise), while 40% of the respondents also thought open source software to be less safe than closed software (a small majority thinks it is better). This means that the open source community in general finds their software to be superior to closed software solutions. Of course the outcomes in this research are clearly biased because of the population that was surveyed.

To determine whether close cooperation between organizations within the open source domain is needed, several questions and propositions were part of the survey. When asked 'Which of the following 6 reasons would get your organization to cooperate with other companies in the open source domain?' more than 75% of the respondents of the survey said their primary reason to cooperate is to exchange information between peers to further improve their software (see figure 2).

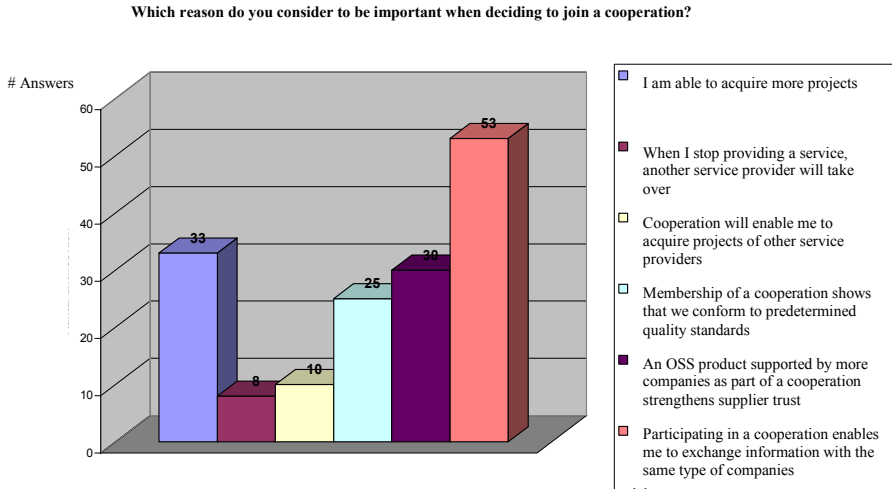


Fig. 2. Reasons to cooperate within the open source domain

The second most important reason (with 47%) is the financial gains organizations hoped to receive in the form of new projects. The continuity of services like maintenance and support that could be guaranteed by cooperating only received support of 11.4% of the respondents and although acquiring new projects is the second most important reason to cooperate not many respondents actually expect to get many new projects (only 14.5% do). The two reasons that have to do with the perceived reliability and quality of an organization by customers get respectively 43% and 35.7% of the respondents approval.

Based on this it seems that the sharing of information (with a focus on software development) is the only trigger to start cooperating. This is to be expected because it is the primary foundation of the open source community. However there is not very much support for cooperation between open source vendors and services suppliers regarding improvement of aspects like maintenance, support and quality. In these topics it seems that organizations in the open source domain stick to their existing business models in which they try to do everything themselves.

The perceived advantages of cooperation (see figure 3) show a similar result. Although financial gains by acquiring projects via cooperation is perceived as an advantage (69%), a majority of the respondents (74%) do not expect their customers to be willing to pay a premium for such a cooperation. Still when they were asked if cooperation within the open source domain could improve the continuity and reliability of support to their customers, 78.5% agreed. When a cooperation takes the form of an organization that is responsible for maintaining quality levels (of member organizations) a large majority of the respondents (91%) stated that the trust in open source and thereby the use would be much higher. In conclusion, we can state that while organizations do think that cooperating is perceived by their customers as an added value, they are not really willing to start such cooperation because they don't

perceive any benefits for themselves and can't direct any costs towards their customers.

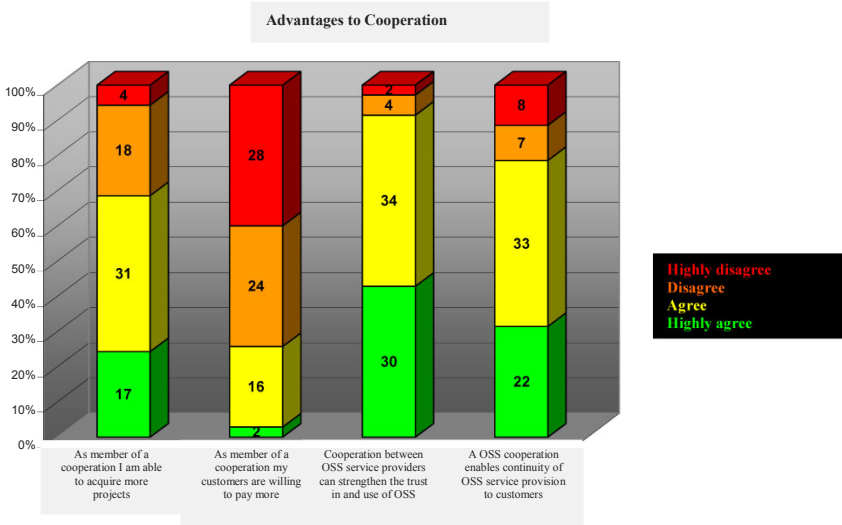


Fig. 3. Perceived advantages of cooperation within the open source domain

5. Discussion and Future Research

This paper describes the outcomes of a multi method research approach to determine if organizations within the open source domain are willing to cooperate. While such cooperation is perceived beneficial there is no positive attitude towards starting such cooperation. However the outcome of this research knows some limitations. First the respondents all are situated in the Netherlands, which makes that the findings may not be applicable to other countries or regions. Second the research is conducted solely at software developers and services providers in the open source software domain; the customers of these organizations have not participated. This means that the perceived value for customers of cooperation as seen by the respondents might be non existent. Finally this research is focused on the willingness to cooperate to improve maintenance, support, reliability and quality of the services and software, other forms of cooperation are out of scope. Therefore the results can't be interpreted as a complete unwillingness of organizations to cooperate with each other. The amount of research done on cooperation between organizations in the open source domain is limited. The findings from this research need further validation at open source users. A next step is research on cultural differences and finally in-depth studies are needed to determine whether perceived advantages and disadvantages are different depending on the type of organization or its maturity.

6. References

- Baarda, DB, de Goede, MPM & Teunissen, J 2001, Basisboek: Kwalitatief Onderzoek (in Dutch), Groningen, Stenfert Kroese.
- Cusumano, M.A. (2004). *The Business of Software*. Free Press. ISBN 0-7432-1580-X.
- Ghosh, R. & Glott, R. (2005). FLOSSPOLs Deliverable D3: Results and policy paper from survey of government authorities [<http://flosspols.org/deliverables/>]
- Goldman, R. & Gabriel R. (2005). Innovation happens elsewhere. Open source as business strategy
- Grand, S., Krogh, G. v., Leonard, D. & Swap, W. (2004). Resource Allocation Beyond Firm Boundaries: A multilevel model for Open Source Innovation. *Long Range Planning* 37 (2004) 591–610.
- Pavlicek, R.C. (2000). *Embracing Insanity. Open Source Software Development*. Sams. ISBN 0-672-31989-6.
- Raymond, E.S. (2001). *The Cathedral and the Bazaar: Musings on Linux and Open Source By an Accidental Revolutionary*. O'Reilly & Associates. ISBN 0-596-00108-8.
- Rijke, H. de (2005). Continuïteit in Open Source diensten. In: *Holland Open*.
- Varian, H.R. & Shapiro, C. (2003). *Linux Adoption in the Public Sector: An Economic Analysis*. [<http://www.sims.berkeley.edu/~hal/Papers/2004/Linuxadoptioninthepublicsector.pdf>]
- Wichmann T. (2002a) Use of Open Source Software in Firms and Public Institutions. Evidence from Germany, Sweden and UK. Berlecom Research GmbH [http://www.berlecon.de/studien/downloads/200207FLOSS_Use.pdf]
- Wichmann T. (2002b) Firms' Open Source activities: motivations and policy implications. Berlecom Research GmbH [http://www.berlecon.de/studien/downloads/200207FLOSS_Activities.pdf]
- Wijnen-Meijer, M. and R. Batenburg (2007). To open source or not to open source: that's the strategic question. Results from a Survey among eight leading software providers. In *Proceedings of ECIS 2007*. St. Gallen, Zwitserland.

Open Source Project Categorization Based on Growth Rate Analysis and Portfolio Planning Methods

Stefan Koch and Volker Stix

Vienna University of Economics and Business Administration

Institute for Information Business

Augasse 2-6

1090 Vienna/Austria

{stefan.koch, volker.stix}@wu-wien.ac.at

Abstract. In this paper, we propose to arrive at an assessment and evaluation of open source projects based on an analysis of their growth rates in several aspects. These include code base, developer number, bug reports and downloads. Based on this analysis and assessment, a well-known portfolio planning method, the BCG matrix, is employed for arriving at a very broad classification of open source projects. While this approach naturally results in a loss of detailed information, a top-level categorization is in some domains necessary and of interest.

1 Introduction

The adoption and evaluation of open source projects has gained increasing interest, both from an academic and business perspective. This has led to the development of assessment schemes like OpenBRR (Open Business Readiness Rating), Open Source Maturity Model, QSOS by Atos Origin, OpenBQR [8] and similar achievements. Most of these approaches are based on detailed scoring of open source products, and aggregation using some form of weightings. While some consider that features of the underlying community form an important part of an evaluation, this is not generally acknowledged. In addition, while in some approaches the use of real data, both on community and the software product itself is planned, some rely on personal rating or data entry of many features.

The approach we will present here is intended to be coupled with a repository of repositories on open source project data [7]. In this context, an enormous amount of data is collected, but this might prove to create additional problems with users of such a service, who might not be able to put it to use. Too much information could effectively hamper their use of the system for rather simple evaluation and adoption tasks or decisions. Therefore, providing a top-level, aggregate classification scheme is necessary. In this paper, we will propose to base such an effort on well-known portfolio planning techniques, especially the BCG matrix. For constructing the axis of such a matrix, several possibilities exist, but in our case we will argue for applying the results of growth rate analysis.

Please use the following format when citing this chapter:

Koch, S. and Stix, V., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succì; (Boston: Springer), pp. 375–380.

2 Growth Rate Analysis

For performing the proposed analysis of open source software projects, the information contained in software development repositories will be used. These repositories contain a plethora of information on the underlying software and the associated development processes [2]. Studying software systems and development processes using these sources of data is very cost-effective and does not influence the software process under consideration. In addition, longitudinal data is available, allowing for analyses considering the project history. Depending on the tools used in a project, possible repositories available for analysis include source code versioning systems, bug reporting systems, or mailing lists.

In open source software development projects, repositories in several forms are also in use, in fact form the most important communication and coordination channels. Therefore only a small amount of information can not be captured by repository analyses because it is transmitted inter-personally. As a side effect, the repositories in use must be available openly and publicly. Therefore open source software development repositories form an optimal data source for studying the associated type of software development. Currently, efforts are underway to consolidate information from diverse sources, in building RoRs (repositories of repositories) [7]. In this pa-per, we propose to build on this infrastructure using appropriate techniques.

For characterising the past development, and also gain an understanding of possible future developments, growth rates can be computed for several aspects like source code, contributors, bug reports, mailing list postings or downloads. All of these might give some insight, while of course the growth in size (of source code) is most often cited (software evolution). For computing and characterising the growth rates, the following methodology will be adopted. This is taken from a prior study of one of the project participants [4] on growth in size.

The first step is to analyse whether a linear or other growth pattern is present in the data. To this end, both a linear and a quadratic model are computed for each project, taking the size in lines-of-code S as a function of the time in days since the first commit t , which is used as project start date, and using one month as time window. Therefore model A is formulated simply as $SA(t) = a * t + b$ and model B as $SB(t) = a * t^2 + t * b + c$. The necessary parameters are to be estimated using regression techniques. As a next step, it is necessary to explore whether the growth rate is decreasing over time. This can be done by analysing the second derivative of the quadratic model $SB(t)'$, or directly the coefficient of the quadratic term a .

The sharp distinction between two groups of projects might prove too inflexible. A new group is therefore introduced representing linear growth in contrast to sub- and super-linear rates. This group is defined as those projects having either a better fit for the linear than the quadratic model, or a coefficient of the quadratic term between -0.1 and 0.1 , thus being very near to zero. This allows for arriving, for each project and each aspect of interest, at a classification for the evolutionary behavior as being either sub-linear, linear, or super-linear.

3 Portfolio Planning and the BCG Matrix

Portfolio planning methods have been applied in strategic decision making for over 20 years [1,9] although they have little theoretical support. They are presented in the literature as diagnostic aids and as prescriptive guides for selecting strategic options [5]. The general idea is to classify positions of products along two dimensions to form a matrix: attractiveness of the market and ability of the product to compete within that market, and to derive insights into strategic actions in this way. Managers often neglect to use a rational economic approach, instead applying un-structured judgmental processes. They may base their decisions on power or emotional factors, which might lead to many of their decisions as being irrational. Thus, portfolio planning methods, such as the BCG matrix, may lead managers to make decisions that are less irrational.

Maybe the most well-known portfolio planning method is the Boston Consulting Group (BCG) method [3], the most widely used portfolio method in US firms [1]. It is based on measuring market attractiveness by market growth rate, and it assesses the firm's ability to compete by its relative market share. The BCG matrix assumes a causal relationship between market share and profitability. It is based on product life cycle theory that can be used to determine what priorities should be given to different products. To ensure long-term value creation, a company should construct a portfolio using products that contains both high-growth products in need of cash inputs and low-growth products that generate cash. Each of the two axes is normally divided into a high and a low portion, resulting in four different quadrants. Each quadrant is assigned both a catching name and a general strategy (see also Figure 1):

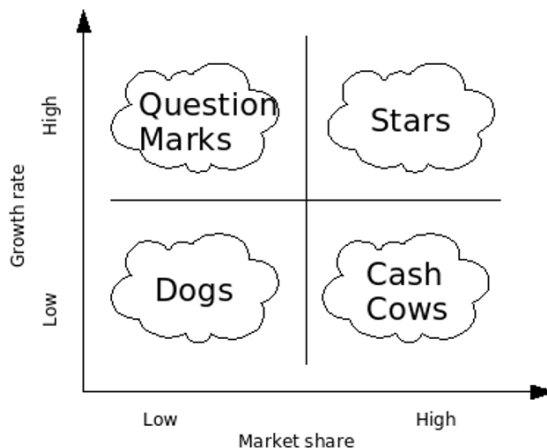


Fig. 1. BCG matrix

Stars are located in the high growth and high market share area. Normally, the cash flow is rather balanced or even, but a position in stars should be maintained. Cash Cows are placed in low growth area coupled with high market share. Profits and cash generation will generally be high, with relatively small investment due to low growth, translating into a very desirable type. Dogs are placed in low growth and low market share quadrant, which are normally associated with a de-invest strategy. Question Marks are enjoying high growth but low market share, resulting in demand for cash but low profits. This kind of product over time might turn into either star or dog, so careful considerations is advised to invest or liquidate.

A review of previously published evaluations of the BCG matrix can be found in [6]. Actual practical use of the BCG matrix is often found to be inhibited by difficulties in measurement of market growth rates and relative market shares. The results are highly sensitive on these measurements. As a result, different matrix methods are likely to yield different recommendations for the same situation.

4 Open Source Matrix and Classification

In this paper, we propose to adopt the BCG matrix approach for classifying open source projects. There are two main aspects to discuss and decide: The construction of the axes, and results of the classification. For constructing and measuring the axes, we propose to use the results of a growth rate analysis. The growth rate of an open source project is constructed using the growth rate in source code (which equals the software evolution viewpoint of software engineering research) and the growth rate of developers. For market share, we propose to use growth rates of bug reports and downloads. Bug reports normally are associated with usage of a product, especially by interested individuals, but might also signal a product with problems.

As each growth rate of the four types used here is classified in one of three steps, conversion into a single measure needed. We propose to use the mean of both rates, with source code respectively downloads taking priority in ties. Using this approach, an open source project matrix can be constructed, with the standard names having been replaced by possible release numbers (see Figure 2).

As can be seen, the classification results in four possible types of projects. This has to be translated into strategies. We need to differentiate between two possible uses, the first one being simple adoption, in which a company or individual wants to decide on which project within a given domain to adopt for a certain task. In the second case, a company wants to build a portfolio of projects. Possible reasons for this include a business model based on a range of software, cooperation with the open source world within a given area, or an application in marketing. Also a company that might pursue a development based on open source, but wants to keep open to several projects might pursue this idea. This leads to the following strategies as-signed to the different types of projects.

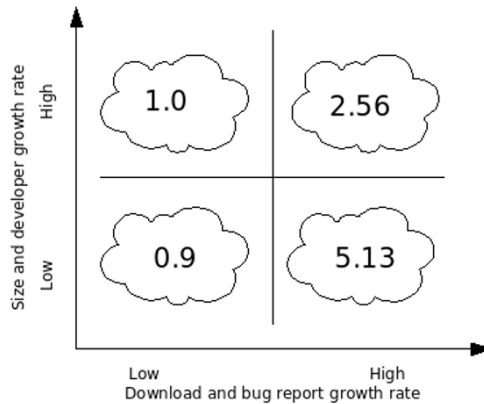


Fig. 2. Open source project matrix

1.0 (Question Marks) currently enjoy huge growth in size and participants, but have not achieved widespread adoption yet. Therefore they might become quite successful, but might fail. For a separate adoption decision, these projects pose considerable risk, while adding 1.0 projects to a portfolio might be interesting.

2.56 (Stars) enjoy both considerable growth and adoption. This makes them interesting candidates for any portfolio selection decision, and candidates for a singular adoption consideration.

5.13 (Cash Cows) have somewhat stabilized in their code and developer growth, but have achieved widespread adoption. This means that normally a mature solution has been found, with less emphasis on introducing new functionalities. This makes these projects prime candidates for consideration for a single adoption decision, and an interesting candidate for portfolio selection. On the other hand, there might be the need for maintenance at a later stage, maybe due to technological changes, but the community is not that active any more.

0.9 (Dogs) have neither huge growth nor adoption, meaning that they prove to be of interest to neither adoption or portfolio considerations. These projects could be termed as failed.

5 Conclusion and Future Research

In this paper we have argued for constructing a top-level open source project classification based on growth rate analyses of several project aspects, and using portfolio planning techniques, especially the BCG matrix. Given current efforts of constructing the databases necessary to support the kind of analyses that form the basis of this approach, pursuing this road seems worthwhile. Portfolio planning approaches have been in business use since several decades, and despite some short-

comings, have provided value and are in wide-spread use. Given that software adoption decisions on organisational levels are often decisions made on a management level, adopting these common approaches might prove beneficial for communication between technical evaluation level and decision authorities.

In future research, an empirical evaluation of the proposed approach would be very important. This would include performing the necessary analyses based on a database, and presenting the results of categorization to decision-makers. There are several possible ways of refinement of the approach that could be pursued: On the one hand, the construction of the axes could be discussed by adding or substituting aspects of projects or communities. On the other hand, there are other portfolio planning approaches besides the BCG matrix that could be explored.

Acknowledgments

This work is partially supported by the Free / libre / open source software metrics and benchmarking study (FLOSSMETRICS), funded by the European Commission under contract number FP6-033982 (www.flossmetrics.org).

References

1. Armstrong, J.S., & Brodie, R.J. (1994) Effects of portfolio planning methods on decision making: experimental results. *International Journal of Research in Marketing*, 11(1):73-84.
2. Cook, J.E., Votta, L.G., & Wolf, A.L. (1998) Cost-effective analysis of in-place software processes. *IEEE Transactions on Software Engineering*, 24 (8):650-663.
3. Day, G. (1986) *Analysis for strategic market decisions*. St. Paul: West.
4. Koch, S. (2007) Software Evolution in Open Source Projects - A Large-Scale Investigation. *Journal of Software Maintenance and Evolution*, 19(6):361-382.
5. Kotler, P. (1984) *Marketing Management*. 5th ed. Englewood Cliffs, NJ: Prentice-Hall.
6. Morrison, A. & Wensley, R. (1991) Boxing up or boxed in? A short history of the Boston Consulting Group share/growth matrix. *Journal of Marketing Management*, 7:105-129.
7. Sowe, S.K., Angelis, L., Stamelos, I., & Manolopoulos, Y. (2007) Using Repository of Repositories (RoRs) to Study the Growth of F/OSS Projects: A Meta-Analysis Research Approach. In *Open Source Development, Adoption and Innovation*, pp. 147-160, Boston: Springer.
8. Taibi, D., Lavazza, L., & Morasca, S. (2007) OpenBQR: a framework for the assessment of OSS. In *Open Source Development, Adoption and Innovation*, pp. 173-186, Boston: Springer.
9. Wind, Y., & Mahajan, V. (1981) *Designing product and business portfolios*.

Applying Open Source Development Practices Inside a Company

Juho Lindman¹, Matti Rossi¹, and Pentti Marttiin²

¹ Helsinki School of Economics, Information Systems Science, PO Box 1210, 00101 Helsinki, Finland. {}@hse.fi, <http://www.hse.fi>

² Nokia Siemens Networks, PO Box 31, 02022 Nokia Siemens Networks, Finland. {}@nsn.com, <http://www.nsn.com>

Abstract. Open Source Software development is seen as a panacea by many companies. The promise of community-style development, innovation and cost savings drive the wider adoption of OSS in companies. However, it is still difficult to institutionalize the open and agile culture of sharing innovation especially into larger departmentalized organizations. The aim of this research paper is to investigate the characteristics of one successful OSS development implementation approach limited inside a company (Inner source). Based on our data, we argue that there are possibilities for employing OSS as a new kind of development process within a company and leveraging thus the innovation potential inside the company.

1 Introduction

It is currently virtually impossible to conduct any business without encountering software created using OSS development methods. Companies are becoming more interested in understanding how to use OSS development inside their organizations or in inter-company development activities [2] [5] [6]. Research indicates that it is possible to build solid business cases around OSS [4].

However, researchers and practitioners are still struggling with the challenge of deploying OSS development practises inside companies. This study aims to provide some initial answers by describing the adoption of inner source portal in a case company. Our viewpoint is that of a support service organization launching new tools for business unit use. The specific case is Nokia, a leading telecommunications company using adopted OSS development process called iSource. We aim to provide an account of the intra-company portal usage and what kind of projects seem to benefit. Findings indicate that 1) there is a growing interest towards iSource, 2) projects that use the portal are heterogeneous, and 3) projects seem to benefit from openness enabled by OSS tooling.

Please use the following format when citing this chapter:

Lindman, J., Rossi, M. and Marttiin, P., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succì; (Boston: Springer), pp. 381–387.

2 Research methodology

Our goal is to provide an account of a successful implementation of OSS practices inside a company. This is necessary because it is widely believed that OSS development will gain more footing in different kinds of organizations [5]. Taking account our goal, we select to conduct an interpretative case study on Nokia iSource service organization [16]. Interpretative approach helped us to take into account the interplay of the service organization and its environment [14]. We also used other methods and sources of evidence to develop our understanding of the events and to ensure the validity of our findings [3].

We started out with managerial interviews and direct key user contacts. Our informants included people from the service organization and business units to build solid case description. The respondents were selected to represent the different levels of inner customers and service organizations. We conducted four interviews and reviewed usage data provided by the technology platform to corroborate the interview findings. Usage data consists of user information, number of projects created and their activity. One of the researchers was working for the company and thus has a very good view of the situation and context of the case from the inside. This might cause bias, but that is offset by the depth gained by being an insider.

As the results from the different data sources seem to be aligned, we assume that the findings support each other. We aimed to make analytical, not statistical generalizations. Statistics offered by the platform did not yet enable developing understanding of models and causal relationships between different data. Thus these two were used in conjunction with interviews and participant observation to validate the case findings.

3 Review of literature

Theories describing OSS phenomenon sometimes suffer from a promoted assumption that OSS communities and processes would be a priori similar [10] [12] [13]. In contrast, it has been shown that OSS communities, software companies and the relations between them are not homogeneous [1]. Thus it seems plausible to presume that also inner source communities are heterogeneous.

The companies that wish to launch communities are in different position than those that wish to benefit from existing OSS software [4]. Also the challenges related to the processes are different depending on whether company has outbound or inbound OSS approach [4]. Commercial context enables different roles for companies using OSS [7] and companies have a variety of choice regarding integrating their operations with the selected business model [9] [11].

Hierarchical organizations benefit from open development communities that promote flow of information and ideas between professionals [15]. In traditional proprietary software development software is developed and maintained centrally in a selected business unit or program. Development community is a way to respond to

the misalignment of experience, skills and interests [8]. This is especially beneficial to large organizations prone to re-organization. Company's priorities might change due to market conditions, often leading to the reallocation of developers into different projects.

Inner Source aims to leverage the power of open source practices, methods and tools in company context [7] [5]. This means in practice: 1) software produced is open to everyone working for a company to use and develop. More accurately, all activities relating to the software development are open and visible to everyone, and 2) interest to develop software initiates a community to take care of the development.

4 Nokia iSource

We selected for analysis a successful OSS inspired implementation, Nokia iSource. Nokia is the world leader in mobile communications. It is a publicly held company with listings in five major exchanges. Today the iSource portal has two instances one for Nokia and another for Nokia Siemens Networks (NSN). NSN is a company created in a merger between Nokia Networks and a part of Siemens in 2007. Our data focuses mainly on events that happened before the merger.

iSource is a company wide portal created in 2001 for managing and working with software assets. Its introduction was originally encouraged by HP's good experiences [2]. Technically, at the time of the study, iSource was a fork of OSS version of Sourceforge portal dated back to 2001 and maintained separately from the mainline. The idea behind iSource was to provide a portal enabling visibility of software and the source code. The goals were to increase individual engineers' awareness of software developed inside the company, and to boost innovation by avoiding the problem of re-implementing the wheel. Currently, projects are counted in hundreds and users in thousands, but activity status varies depending on the project life-cycle and funding. iSource has a support service, that promote iSource service and champions its adoption among business.

5 Observations from cases

5.1 Increase in adoption

According to the user and project statistics iSource has enjoyed stable increase in use. This trend can be observed from Figure 1 below, which shows the number of new projects created per year (the exact numbers were omitted to protect company confidential information, but the trend is visible).

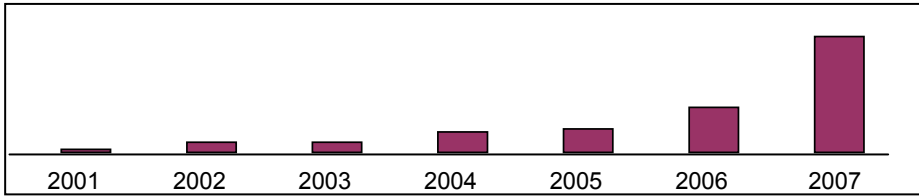


Fig. 1. New iSource projects with their creation year

5.2 Project heterogeneity and openness

iSource has been used for different kinds of projects. This variety can be observed for example when comparing the number of developers and the administrators or the ratio of the two. Figure 2 shows the numbers of developer per project and Figure 3 the number of administrators.

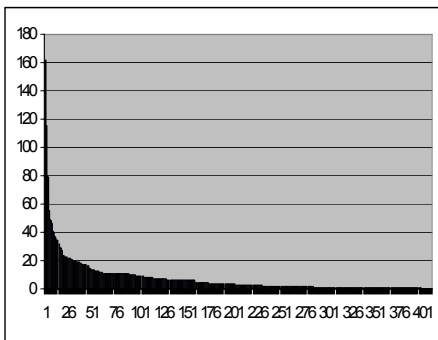


Fig. 2. The number of developers/project

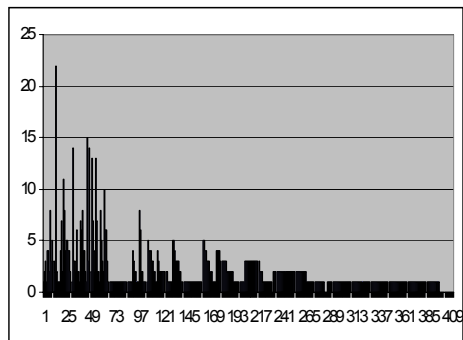


Fig. 3. The number of admins/project number

Most projects using iSource are having one or more of the following characteristics:

1. Projects have selected Subversion or CVS (Concurrent Version System) as their SCM tool (iSource service is the official one for hosting these SCM repositories).
2. Projects are distributed to multiple sites and continents (open source tools require less bandwidth)
3. Projects are using agile development methods (OSS tooling attracts agile projects) or take advance of continuous integration tools (e.g. CruiseControl or Bamboo)
4. Software is developed in inter-company collaboration (iSource is opened to collaborators)

5. Projects are advancing Inner Source (openness when sharing e.g. scripts and libraries; OSS development model in platform development)

The following table shows three top-10 lists of iSource projects. It was what kinds of projects have most developers, are benefitting of file downloading, and are using CVS. The project identification (or categorization) and is done by selecting a key word from the project's public summary.

Table 1. Breakdown of top-10 lists: # of developers, file downloads and CVS usage (commits).

Categorization (Most developers)	#	Categorization (File Downloads)	#	Categorization (CVS commits)	#
Platform	6	Framework	3	Platform	4
Simulator	2	Scripts and libraries	2	Framework	2
Reference Implementation	1	SDK	1	Emulator	1
Testing environment	1	Software suite	1	Simulator	1
		Platform	1	Language	1
		Emulator	1	Protocol architecture	1
		Language	1		

First, 10 projects with the biggest developer number were overviewed. It seemed that among the biggest projects there were several platform projects as indicated in Table 1. Not all the projects are ranked in downloads list. This probably means that no files are stored. The projects on this list are different from those that have the most developers. Smaller projects are able to generate interest and share for example emulators, scripts and libraries. Downloads also correlate with project age. Projects may use the portal with or without SCM. Based on our statistics less than 50% of projects have used or at least tried CVS (Subversion was just introduced by the service). Some projects in our table 1 have also been transferred to Sourceforge at some point of their life-cycle. Two open sourced projects (Sofia SIP <http://sourceforge.net/projects/sofia-sip/> and Python for S60 <http://sourceforge.net/projects/pys60/>) are the top of the iSource lists.

6 Findings and discussion

The aim of our research was to describe usage and show what kind of projects benefit from intra-company portal. There is a clear growing trend in iSource usage. As projects adopted it voluntarily, there is clearly an expectation of benefit. The portal launch needs to be planned, projects need support and there needs to be a robust technical support and evangelists to keep the platform alive.

Projects inside iSource are heterogeneous: most of the biggest projects are platform projects, but on the other hand frameworks and libraries are more downloaded. So, we cannot claim homogeneity between projects. One way to characterize the different project types would be to divide them into: 1) SCM projects, 2) distributed projects, 3) agile projects, 4) intra-company collaboration projects, and 5) Inner Source projects utilizing the full potential of OSS approach.

Openness seems to support adoption of the project results. Not only have projects been able to gain visibility inside the company, but some iSource projects have been moved into open domain once tested inside the global company.

When describing one case study, there is always a question of how to transfer the findings into other companies. This is especially true when dealing with global company tool adoption. However, from our industrial experience and discussions with several other companies in scope of a European research project, it seems that similar challenges need to be solved when moving towards developer communities. The experience of the companies already implementing inner source are very valuable to other industry actors considering moving towards similar development practices.

Acknowledgements

The authors would like to thank the ITEA-COSI project.

References

1. L. Dahlander, M. Magnusson. Relationships Between Open Source Companies and Communities: Observations from Nordic Firms. *Research Policy*, 34(4): 481-493, 2005.
2. J. Dinkelacker, P. Garg, R. Miller, D. Nelson. Progressive Open Source. *Proceedings of the 24th International Conference on Software Engineering, Buenos Aires, Argentina, May 2002*.
3. K. M. Eisenhardt. Building Theories from Case Study Research. *Academy of Management Review* 14 (4): 532-550, 1989.
4. M. Fink. *The Business and Economics of Linux and Open Source*. Prentice Hall, New Jersey, 2002.
5. B. Fitzgerald. The Transformation of Open Source Software. *MIS Quarterly*, 30(2):587-598, 2006.
6. V. Gurbani, A. Garvert, and J. Herbsleb. A Case Study of Open Source Tools and Practices in a Commercial Setting. In *Open Source Application Spaces: Proceeding of the Fifth Workshop on Open Source Software Engineering, St Louis, USA, 17-21 May 2005*.
7. Ø. Hauge, C. Sørensen, and A. Røsdal. Surveying Industrial Roles in Open Source Software Development. *OSS 2007. June 11-14 2007, Limerick, Ireland*.
8. C. Melian, CB. Ammirati, P. Garg, and G. Sevon. Building Networks of Software Communities in a Large Corporation. Hewlet Packard POS, 2001.

9. A. Osterwalder, Y. Pigneur. Clarifying Business Models: Origins, Present and Future of the Concept. *Communications of the AIS*, v. 15, May 2005.
10. B. Perens. The open source definition. In C. DiBona, S. Ockman, and M. Stone, eds, *Open Sources: Voices from the Open Source Revolution*. O'Reilly & Associates, Sebastapol, CA, 1999.
11. R. Rajala, M. Westerlund, and J. Nissilä, *Software for Free? Revenue Models In The Open Source Software Business*. Proceedings of the 5th Global Conference on Business&Economics, Cambridge University, Cambridge, UK, 2006.
12. E.S. Raymond. *The Cathedral and The Bazaar – Musings on Linux and Open Source by Accidental Revolutionary*, O'Reilly&Associates, Sebastopol, CA., 1999.
13. A.M. Szczepanska, M. Bergquist and , J. Ljunberg. High Noon at OS Corral: Duels and Shoot-Outs in Open Source Discours. In: J. Feller, B. Fitzgerald., S. A. Hissam, and K. R. Lakhani, eds. *Perspectives on Free and Open Source Software*, MIT Press, Cambridge, Mass., 2005.
14. G. Walsham. *Interpreting Information Systems in Organizations*. Cichester, Wiley&Sons. 1993.
15. G. von Krogh, E. von Hippel. The Promise of Research on Open Source Software. *Management Science*, 52(7): 975-983, July 2006.
16. R. K. Yin, *Case Study Research, Design and Methods*, 2nd ed., Sage Publications, Newbury Park, 1994.

Towards The Evaluation of OSS Trustworthiness: Lessons Learned From The Observation of Relevant OSS Projects

Davide Taibi¹, Vieri del Bianco¹, Davide Dalle Carbonare²,
Luigi Lavazza¹, and Sandro Morasca¹
¹ University of Insubria

davide.taibi | vieri.delbianco | luigi.lavazza | sandro.morasca@uninsubria.it

WWW home page: <http://www.uninsubria.it>

² Engineering Ingegneria Informatica S.p.A.

davide.dallecarbonare@eng.it

WWW home page: <http://www.eng.it>

Abstract. To facilitate the adoption of open-source software (OSS) in industry, it is important to provide potential users (i.e., those who could decide to adopt OSS) with the means for evaluating the trustworthiness of OS products. This paper presents part of the work done in the QualiPSo project for this purpose. A set of factors that are believed to affect the perception of trustworthiness are introduced. In order to test the feasibility of deriving a correct, complete and reliable evaluation of trustworthiness on the basis of these factors, a set of well-known OSS projects have been chosen. Then, the possibility to assess the proposed factors on each project was verified: not all the factors appear to be observable or measurable. The paper reports what information is available to support the evaluation and what is not. This knowledge is considered to be useful to users, who are warned that there are still dark areas in the characterization of OSS products, and to developers, who should provide more data and characteristics on their products in order to support their adoption.

Keywords: OSS trustworthiness, OSS quality, OSS adoption

1 Introduction

The success of OSS is due to multiple reasons, ranging from technical qualities to financial, ethical and political motivations. Nonetheless, the adoption of OSS is still limited. The reason is that, in several cases, OSS fails to convince potential users that its adoption is safe and poses no more risks than purchasing commercial software. In this paper, we report on the initial work, carried out in the QualiPSo project, that focuses specifically on the characteristics of OSS products and artefacts, in order to identify the ones most closely related to trustworthiness. The QualiPSo (Quality Platform for Open Source Software) project 0 is an ongoing initiative that proposes a

Please use the following format when citing this chapter:

Taibi, D., del Bianco, V., Carbonare, D.D., Lavazza, L. and Morasca, S., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 389–395.

coherent and systematic evaluation of the trustworthiness of OSS projects, and aims at promoting the diffusion of OSS by focusing on OSS trustworthiness.

We name “trustworthiness” the set of qualities that are of interest for the users, especially in the process of deciding whether a given OS program (or library, or other piece of software) is “good enough” to be used in an industrial or professional context.

Firstly, we defined the set of factors that were believed to be the most closely related to the perceived trustworthiness 0; then we identified a set of OSS projects, widely adopted and generally considered trustable, to be used as references. Afterwards, a first quick analysis was carried out, checking which factors were readily available on each project’s web site. The idea was to emulate the search for information carried out by a potential user, who browses the project’s web sites, but is not willing to spend too much effort and time in carrying out a complete analysis. Since the view of trustworthiness factors emerging from the analysis seemed too subjective, it was decided to precisely define measures specifying how to evaluate the OSS characteristics, and how to collect data that could be effectively used in the analysis phase, to be performed according to some statistical methods.

2 Project Selection and Analysis

The selection of projects addressed different types of software applications, generally considered stable and mature. The complete set of projects comprises 32 products, different with respect to age, implementation language, size of developers and users communities, etc.

Here the criteria used to select a representative set of OSS projects are reported. Projects have a set of characterising attributes. The selection criteria aimed at:

- Including a reasonably small set of projects.
- Including at least a couple of projects for every possible value of any attribute.

For instance, an attribute is the size of the development team. Four possible values were defined: 0 (inactive project), no more than ten people, up to 50 people, more than 50 people. Therefore, we took care to include at least two projects for each of the four mentioned classes. The complete set of attributes is reported in Table 1.

Table 1. The projects’ attributes.

Attribute	Possible values
Repository	SourceForge, Apache, Java.net, FreshMeat, Rubyforge, ObjectWeb, Free Software Foundation, SourceKibitzer, other
Standalone	Yes/No (Part of a Project family)
Type	Web Server, Operating System, ERP, CSM, ...
Developer organization type	Sponsored/foundation, spontaneous, other
Cost	Free; pay for services and features; pay for everything
Size of the development team	0 (abandoned/closed project), 1–10, 11–50, >50

User community size	Small (<51), Medium (51–250), Large (>250)
Programming language	Java, C#, C/C++, scripting languages, Visual Basic, other
Tool support(*)	little use of tools (0-4 tools used); extensive use of tools (5-7)
Innovation	Traditional application (existing before 2003); Emerging application (only proprietary solutions before 2003)
Age	Project started before 1998; between 1998 and 2003; after 2003

(*) the potentially supported activities are: continuous integration (supported by Cruise Control, Damage Control, Continuum, ...), building (Ant, make, ...), code documentation (Doxygen, Javadoc, ...), testing (JUnit/NUnit/PHPUnit, ...), version control (CVS, Subversion, ...), bug tracking (Jira, Bugzilla, ...), static code analysis (Spoon, Checkstyle,...).

We analyzed the 32 selected projects, considering the information concerning the trustworthiness factors. The analysis was carried out by looking for information that was readily available in the project web sites.

Table 2 lists the factors that are believed to determine trustworthiness, and reports, for each factor, how many projects provided (in the official web site) enough information to evaluate the factor.

Table 2. Number of projects that provide data about the considered factors

Factor	N° of projects supporting the evaluation of the factor
Type of licenses used	32
Facilities for developing modifying...	10
Technical manual	2
Mid/long term existence of a user community	Not Found
Existence of a sufficiently large community	Not Found
Short term support	Not Found
Mid/long term existence of a maintainer organization	Not Found
Use of standard architecture	11
Usage of patterns	6
Programming language uniformity	25*
Complexity	Not Found
Size	Not Found
Reliability	8
Maintainability	7
Modularity	18
Usability	9
Portability	15
Performances	8
Functional requirements satisfaction	11
Customer satisfaction	Not Found
Interoperability with external systems/integration ease	10

Availability of user manual	30
Standard compliance	10
Availability of best practices	17
Human interface language/localization	15
Self containedness	15
Existence of benchmarks or test suites that witness quality	5
Availability of training, guidelines, use cases, tutorial	20
Distribution channel	31
Number of downloads	32

* Only a few of these projects explicitly declare to use one language for the whole project.

3 Factor Refinement

The experience of the quick project analysis showed that for several factors it was necessary to define more precise and specific measures. The need to base evaluations on more objective data also emerged. Accordingly, whenever a factor proved not to be directly measurable, a set of ‘proxies’ was defined. Some proxies can be assessed in a simple and direct manner, while others need specific tools. Table 3 reports both the new measures and the unchanged ones defined for OSS product trustworthiness. The idea is that for each OSS project the factors are evaluated according to these measurement definitions.

Table 3. New criteria for the evaluation of trustworthiness factors

Factor	Basic indicators
Functional requirements satisfaction degree	Availability of: feature list, free text description, release notes, product example/demo
Customer Satisfaction	List of organizations, testimonials and other projects using this software, case studies, usage histories User community satisfaction (according to forums, blogs, mailing lists, newsgroups, magazine/scientific articles)
Interoperability	Communication with other systems supported by suitable mechanisms (SOAP, Web services, protocols, public interfaces, ...); Ease of integration with other products and possibility to migrate to other product with little effort
Reliability	Development status, frequency of patches, average bug time solving
Maintainability	Existence of a guide to extend/adapt the OSS product, maintenance releases and architectural documentation. Coherent usage of coding guidelines/standard, source code quality and programming language uniformity
Modularity	The product provides plug-in interface(s)
Standard Architecture	Availability of architectural documentation and usage of architectural standard/pattern

Mid Long Term Existence of a User Community	Project Age; Trend of the number of users; Number of patches/releases in the last 6 month; Number of developers involved; Average bug solving time
Availability of technical and user documentation	Availability of: up to date technical/user manual, getting started guide, installation guide, Technical/User related F.A.Q., Technical/user forum and mailing list
Standard Compliance	Any information about standard implemented (like HTTP 1.0, SQL 97...) and coding standards
Existence of a sufficiently large community of users	Number of posts available on forums/blogs/newsgroups and related activity
Performance	Existence of performance tests and/or scenarios, specific performance-related documentation Implementation -Any best practices, concerning design and product construction, aimed at boosting performance.
Type of License	Main and sub license used
Short Term Support	Bug number, bug removal rate, availability of professional services
Availability of facilities for developing, modifying, and customizing the product	General purpose build tools applicable to the product, build script, built-in customization facility (configuration API, ...)
Usability	Detailed feature description and user manual Ease of installation/configuration, ease of use.
Portability	Supported environments, usage of a portable language (like Java), environment-dependent implementation (e.g., usage of hardware/software dependent libraries)
OSS Provider Reputation	Opinion and feedback from other users
Best Practices	Availability of best practices, code examples/tutorials
Programming language uniformity	Number of languages used in the project
Complexity	McCabe complexity number or any related information available on the web site
Human Interface Language Localization	Localization support availability (e.g., are language files provided?)
Self Containedness	Can the product be installed and executed "out of the box" or does it require other software? Are dependencies documented?
Existence of benchmarks/test suites that witness for the quality	Availability of test suites/benchmarks, Usage of a test framework (JUnit, DejaGNU,...), results of tests published (on the project site), existence of initiatives to encourage the community to contribute to quality efforts
Mid/long term existence of a maintainer / sponsor	Active maintainer organization / sponsor
Availability of training, guidelines, use cases, tutorial etc.	Up to date training materials, manuals and guidelines available free of charge Availability of official training courses
The distribution media	Source code download; Binaries download Access to the project repository; CD/DVD distribution
Size	Number of Lines of code, source files and functions (or classes

	and methods, for object oriented code)
Popularity of the product	Number of downloads

The main areas that could not be covered in the previous analysis were those related to the quality of the product and the user community. In no project website we found any indication about the user community size, the internal software quality and complexity, or the vitality of the project.

Proxy definitions were conceived to make the assessment as easy and objective as possible. The possibility of employing tools –possibly specifically developed for this purpose– was also taken into account. Considering for instance the evaluation of *the mid/long term existence of a user community*, we suggest checking the following indicators: the growth rate of the community, the number of patches/releases during the last 6 months, the number of developers involved, the average bug solving time and the project age. In order to assess the growth trend of the user community we shall develop a specific tool, which –by digging into the web portals, official forums and blogs– computes the number of unique users in the last months.

Unfortunately, some information considered important is never exposed on the project websites; therefore we wish to throw a suggestion to the Open Source community, recommending the leaders of OSS projects who would like to publicize the trustworthiness of their products to publish all the useful data. In any case, there are some factors that are inherently difficult to evaluate: for instance, it is quite hard to evaluate the quality of the supplied user manual. This task could be made much easier if it were possible to collect feedback from users: it is thus important to make the users aware that the usage of some feedback collector would be beneficial to the whole user community.

4 Conclusions

In order to favour the adoption of OSS, it is necessary to assure the potential users that the OSS products do meet their expectations under several respects. In the QualiPSo project, the notion of “trustworthiness” is meant to include several qualities of the OSS, ranging from training support to the possibility of modifying/adapting the programs, to the availability of support from the producer, etc. Since the notion of trustworthiness is quite broad, it is necessary to fully understand what factors contribute to making a product trustworthy. For this purpose, in the QualiPSo project several OSS products have been examined: a set of factors that are believed to affect the perceived trustworthiness were identified, a set of outstanding OSS project were selected and are being analysed with respect to the mentioned factors.

Here we reported the preliminary results of the analysis of OSS products and artifacts. Next steps will include a second analysis round, based on the usage of the newly defined measures, possibly along with a campaign, addressed to OSS

developers, to provide more information on the characteristics that affect the trustworthiness of their products. A number of OSS code repositories will also be analyzed, with the aid of automated tools.

The collected data will be analyzed, with the objective of identifying commonalities and differences in the characteristics and usage of OSS products, to prepare the ground for the creation of a trustworthiness model encompassing the characteristics and factors that have been observed to actually affect the perception of trustworthiness in OSS products and artifacts.

Although the analysis is not yet complete, we were able to make some preliminary observations. A first result is that by browsing the information provided by the projects' web sites, only a relatively small set of the interesting factors could be evaluated: several factors appear not to be observable or measurable. The paper reports what information is not available: developers should provide the missing information in order to support the adoption of their products. Since some of the trustworthiness factors could not be evaluated because of their subjectivity, we began a more precise definition of the measures – illustrated in Table 3 – that should be used to capture these factors.

References

1. V. del Bianco, M. Chinosi, L. Lavazza, S. Morasca, D. Taibi, "How European software industry perceives OSS trustworthiness and what are the specific criteria to establish trust in OSS", October 2007, available on-line at <http://qualipso.semanticdesktop.org/xwiki/bin/view/Wiki/WP51>.
2. QualiPSo project web site: <http://www.qualipso.eu>

Open Source Reference Systems for Biometric Verification of Identity

Aurélien Mayoue and Dijana Petrovska-Delacrétaz
TELECOM & Management SudParis,
9 rue Charles Fourier,
91011 Evry Cedex, France

Abstract. This paper focuses on the common evaluation framework which was developed by the BioSecure Network of Excellence during the European FP6 project BioSecure (Biometrics for Secure authentication). This framework, which is composed of open-source reference systems, publicly available databases, assessment protocols and benchmarking results, introduces a new experimental methodology for conducting, reporting and comparing experiments for biometric systems, participating to standardisation efforts. Its use will permit to make a link between different published works. It provides also the necessary tools to assure the reproducibility of the benchmarking biometric experiments. This framework can be considered as a re-liable and innovative process to evaluate the progress of research in the field of bio-metrics.

1 Introduction

With the widespread use of biometric systems in our society, a lot of articles about this topic have been published in the last years. Among all the algorithms presented in the literature, it seems essential to know which one performs better, hence the need for a common evaluation framework which would enable a fair comparison between biometric algorithms. Indeed, the performance of two biometric verification systems can only be fairly compared in the case where these both systems have been evaluated in the same experimental conditions, i.e. applied on the same database according to the same protocol. That's why several biometric evaluation campaigns have been launched during the last years. But in the existing ongoing evaluation programs, such as NIST-SRE [1] for speaker recognition, BANCA [2], M2VTS [3] and FRVT [4] for face, FVC [5] for fingerprint, SVC [6] for signature and ICE [7] for iris, the databases used are often not publicly available and there is no systematic usage of a state-of-the-art algorithm to calibrate both the performance of the evaluated systems and the 'difficulty' of the evaluation database and protocol. Therefore, it is impossible to reproduce the experiments and to measure the progress (in comparison with the current state of the art) done by the newly developed research prototypes which are tested during these evaluation campaigns. In this way, such evaluations can not be considered as an universal evaluation tool.

Please use the following format when citing this chapter:

Mayoue, A. and Petrovska-Delacrétaz, D., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 397–404.

From these observations, the partners of the BioSecure Network of Excellence (NoE) [8] have brought together the expertise in a wide range of biometric modalities in order to make available to the biometric community a new open-source evaluation tool, which enables a fair comparison of the biometric algorithms being tested. In this way, the NoE has taken in charge the development or improvement of existing open-source identity verification systems (softwares) for the most widely used biometric modalities (2D and 3D faces, fingerprint, hand, iris, signature, speech and talking-face). These open-source systems are based on principles that have proven their efficiency and their purpose is to serve as comparison point to measure the real progress achieved with new research methods. Consequently, these open-source systems are known as reference systems. These systems accompanied with benchmarking (calibration) publicly available databases and benchmarking (calibration) assessment protocols define the building blocks of the evaluation framework, which also reports benchmarking results (i.e. results obtained by applying the reference system on the benchmarking database according to the benchmarking protocol).

From now on, any research laboratory can evaluate their own biometric algorithms using the benchmarking database in accordance with the benchmarking protocol and then compare their results to the performance of the reference system. This is a new way to calibrate the performance of an algorithm.

The outline of this paper is as follows. In Sect. 2, the objectives of the proposed evaluation framework are developed. Then, the open-source reference systems, which compose this framework, are briefly presented in Sect. 3 whereas in Sect. 4 we explain how to use the evaluation tool in practice. Section 5 ends up with conclusions and perspectives.

2 Objectives

2.1 Properties of the evaluation framework

In the framework of identity verification, it is very difficult to compare two different methods from two different articles in the literature, even though they deal with the very same task. It raises a real problem when one wants to know if a new original method performs better than the current state of the art for example. This can be explained by the fact that a lot of researchers have recorded their own test data-base and are the only one performing experiments on it, which are subsequently impossible to reproduce.

Our evaluation framework brings an easy yet efficient answer to this problem. As it is composed (for each modality) of a reference system, a publicly available data-base and an evaluation protocol accompanied with a documentation which reports performance of the reference system, our evaluation framework can be used in two ways:

- in the case where someone publishes results on a specific database according to a specific protocol, experiments using the reference system have to be added as a way of calibrating the difficulty of this particular task.
- in the case where someone publishes results using the benchmarking data-base and protocol, the benchmarking results of the reference system can be used as a way of calibrating the performance.

In both cases, the results of the reference systems permit to measure the progress achieved with the newly developed algorithm being tested. Now, it is hoped that the published results of a new biometric algorithm will be systematically accompanied with the results obtained by the reference system in the same experimental conditions. Such a methodology for reporting experimental results will facilitate deeply advances of the state of the art.

Through our framework, an absolute reproducibility of the benchmarking results is also assured. Indeed, an explanatory documentation is available for each modality to avoid any possible misinterpretation of the proposed evaluation framework. In this way, the properties of our framework are the following:

- its universal use: for each modality, it is comprised of an open-source reference system, a publicly available database and a well defined evaluation protocol.
- it enables a fair comparison of the biometric algorithms being tested.
- it evaluates the real progress achieved with new research prototypes.
- its easy use: for each modality, a document explains in details the evaluation protocol and how to install and use the reference system.
- the full reproducibility of the benchmarking results.

2.2 Relevance of using open-source reference systems

The fact that the proposed reference systems are open-source and made of replaceable C/C++ software modules with well defined inputs/outputs is of great interest. Indeed, researchers often work on a specific part of the system and do not have the time nor the interest in building a complete system. In this way, a researcher could show the improvement of his new features extraction algorithm (for example) simply by replacing the corresponding module in the reference system and without having to bother about the pattern recognition algorithm. This new methodology for conducting research and experiments will enable to assess the contribution of each module to the final performance of a biometric system.

Using an open-source reference system as a basis for researching a specific area is also a good way to save time, human resources and money. For example, young researchers who are just starting in the field of biometrics, could immediately be faced with the reference system and they could put all their efforts to improve our efficient system rather than developing a state-of-the-art algorithm from A to Z again. In this way, the main advantages of our reference systems are the following:

- development as open-source software.

- easy implementation: in general, a reference system is composed of four replaceable modules (pre-processing, features extraction, model building and matching).
- time saving: a reference system can be considered as a starting point for the development of a more efficient system.

3 The Open-source Reference Systems

At time of writing, the evaluation framework concerns eight biometric modalities. It is comprised of five publicly available databases and ten benchmarking open-source reference systems. Among these systems, eight were developed within the framework of the BioSecure NoE and only three were existing ones (the fingerprint and speech reference systems) but all systems are accompanied with a benchmarking database, assessment protocol and benchmarking results to compose our evaluation framework:

1. For the 2D face modality, the BioSecure reference system is based on the Eigenface approach [9] and the BANCA database [10] is chosen for the benchmarking experiments.
2. For the 3D face modality, the BioSecure reference system [11] uses both Iterative Closest Points [12] and Thin Plate Spline warping [14] algorithms. The associated database is the 3D RMA database [13].
3. For the fingerprint modality, the NIST reference system (NFIS2 [15]) uses a standard minutiae based approach. MCYT-100 [16] is selected as the benchmarking database.
4. For the hand modality, the BioSecure reference system [17] uses geometry of fingers for recognition purposes. The selected database is the Biosecure database [17].
5. For the iris modality, the BioSecure reference system (OSIRIS [18]) is deeply inspired by Daugman's works [19] and the database used for the benchmarking experiments is CBS [20].
6. For the signature modality, two reference systems developed by BioSecure members are available. The first one [21] is based on the Hidden Markov Model [22] whereas the second one [23] uses the Levenshtein distance [24]. The MCYT-100 [16] database is chosen for the benchmarking experiments.
7. For the speech modality, the evaluation framework is composed of two open-source softwares: ALIZE [25] and BECARS [27]. Both systems are based on the Gaussian Mixture Models approach [26]. The reference data-base is the speech part of the BANCA database [10].
8. For the talking-face modality, the BioSecure reference system [28] is based on the fusion of face and speech scores. The BANCA database [10] is used for the benchmarking experiments.

Some of the open-source reference systems available in the benchmarking framework have been already used in major existing evaluation campaigns. For example, the ALIZE software [25] is consistently used during the NIST-SRE [1] and it exhibits state-of-the-art performance. The iris reference system [18] has been tested with the ICE'2005 database and it has a quite good ranking compared to the results re-reported in [7]. Endless, the signature reference system based on HMM [21] has obtained good performance at the SVC'2004 competition [6]. More details about performance of all reference systems can be found in [29].

4 Use of the Evaluation Tool

The source code of the ten reference systems described in Sect. 3 are available at [30]. Scripts (required to run a full experiment with the reference system), lists of tests (impostor and genuine trials) to be done and all the necessary information needed to fully reproduce the benchmarking results are also available at this address. Endless, for each modality, a document explains how to install and use the reference system. In practice, the user of our evaluation framework can be confronted with three different scenarios:

1. He wants to evaluate his new biometric algorithm using our reference data-base and protocol to calibrate the performance of his algorithm. In this case, he has to process as follows:
 - run the evaluated system on the reference database using the list of trials provided within the evaluation framework.
 - compare the obtained results to the benchmarking results provided within the evaluation framework to measure the progress achieved with the new algorithm.
2. He wants to test the reference system on his own database according to his own protocol in order to calibrate this particular task. In this case, he has to process as follows:
 - run the reference system using his own database and protocol.
 - compare the obtained results to those provided within the evaluation framework to evaluate the 'difficulty' of these particular database and protocol.
 - use the obtained results as comparison point for the other experiments carried out in the same conditions.
3. He wants simply to reproduce the benchmarking results to be convinced of the good installation and use of the reference system:
 - run the reference system on the reference database using the scripts and lists of trials provided within the evaluation framework.
 - verify that the obtained results are the same as the benchmarking results.

Since the reference systems are open-source, other scenarios are of course conceivable. Some parameters or even some modules of the reference systems can be changed in order to improve their basic performances.

5. Conclusions and Perspectives

In the field of biometry, the BioSecure NoE has developed an universal evaluation framework based on open-source reference systems, publicly available data-bases, evaluation protocols and benchmarking results. This framework enables a fair comparison of the performance of biometric systems and its widespread use would facilitate the advances of the biometric state of the art.

This framework has been used for the first time during the BioSecure Multimodal Evaluation Campaign (BMEC) 2007. The organizers of this competition have systematically used the reference systems to calibrate the ‘difficulty’ of the newly constituted BioSecure database and protocol. Then, the results of the reference systems have been used for measuring the improvements acquired with each submitted re-search system. All results of this evaluation are available on the BMEC’s web site [31].

As the research advances, the reference systems should be updated in order to remain competitive. The development of reference systems for the newly appeared modalities (such as palm vein) is also part of our future plans.

Acknowledgments

Biosecure is a project of the 6th Framework Programme of the European Un-ion. We thank in particular all the universities which participated in the development of the BioSe-cure reference systems.

References

1. NIST Speaker Recognition Evaluations. <http://www.nist.gov/speech/tests/spk>.
2. BANCA: Biometric Access control for Networked and e-Commerce Applications. <http://www.ee.surrey.ac.uk/CVSSP/banca/>.
3. M2VTS: Multi Modal Verification for Teleservices and Security applications. <http://www.ee.surrey.ac.uk/CVSSP/xm2vtsdb/>.
4. FRVT: Face Recognition Vendor Test. <http://www.frvt.org/>.
5. FVC: Fingerprint Verification Competition. <http://bias.csr.unibo.it/fvc2006/>.
6. SVC: Signature Verification Competition. <http://www.cse.ust.hk/svc2004/>.
7. ICE: Iris Challenge Evaluation. <http://iris.nist.gov/ice/>.
8. BioSecure Network of Excellence. <http://biosecure.info/>.

9. M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71-86, 1991.
10. E. Bailly-Baillière, S. Bengio, F. Bimbot, M. Hamouz, J. Kittler, J. Mariéthoz, J. Matas, K. Messer, V. Popovici, F. Porée, B. Ruiz, and J.-P. Thiran. The BANCA Database and Evaluation Protocol. In 4th International Conference on Audio-and Video-Based Biometric Person Authentication (AVBPA'03), volume 2688 of *Lecture Notes in Computer Science*, pages 625-638, Guildford, UK, January 2003. Springer.
11. M.O. Irfanoglu, B. Gokberk, and L. Akarun. 3d Shape-based Face Recognition Using Automatically Registered Facial Surfaces. In *Proc. 17th International Conference on Pattern Recognition*, Cambridge, UK, 2004.
12. P. Besl and N. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239-256, 1992.
13. 3D RMA database. http://www.sic.rma.ac.be/~beumier/DB/3d_rma.html.
14. F.L. Bookstein. Principal Warps: Thin-Plate Splines and the Decomposition of Deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:567-585, 1989.
15. C. Watson, M. Garris, E. Tabassi, C. Wilson, R. McCabe, and S. Janet. Guide to NIST Finger-print Image Software 2 (NFIS2) - <http://fingerprint.nist.gov/NFIS>.
16. J. Ortega-Garcia, J. Fierrez-Aguilar, D. Simon, M.F.J. Gonzalez, V. Espinosa, A. Satue, I. Hernaez, J. J. Igarza, C. Vivaracho, D. Escudero, and Q. I. Moro. MCYT baseline corpus: A bi-modal biometric database. *IEE Proceedings Vision, Image and Signal Processing, Special Issue on Biometrics on the Internet*, 150(6):395-401, December 2003.
17. G. Fouquier, L. Likforman, J. Darbon, and B. Sankur. The Biosecure Geometry-based System for Hand Modality. In the *Proceedings of 32nd International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Honolulu, Hawaii, USA, april 2007.
18. E. Krichen. Reconnaissance des personnes par l'iris en mode dégradé. Thèse de doctorat, Institut National des Télécommunications, 2007.
19. J.G. Daugman. High confidence visual recognition of persons by a test of statistical independence. *IEEE Trans. Patt. Ana. Mach. Intell.*, 15(11):1148-1161, 1993.
20. CASIA iris image database. <http://www.cbsr.ia.ac.cn/IrisDatabase.htm>.
21. B. Ly Van, S. Garcia-Salicetti, and B. Dorizzi. On using the Viterbi Path along with HMM Likelihood Information for On-line Signature Verification. In *Proceedings of the IEEE Transactions on Systems, Man and Cybernetics, Part B*, to appear.
22. L. Rabiner and B.H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall Signal Processing Series. 1993.
23. S. Schimke, C. Vielhauer, and J. Dittmann. Using Adapted Levenshtein Distance for On-Line Signature Authentication. In *Proceedings of the ICPR 2004, IEEE 17th International Conference on Pattern Recognition*, ISBN 0-7695-2128-2, 2004.
24. V.I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics*, 10:707-710, 1966.
25. ALIZE software. <http://www.lia.univ-avignon.fr/heberges/ALIZE/>.
26. D.A. Reynolds. A Gaussian Mixture Modeling Approach to Text-Independent Speaker Identification. Ph.D. Thesis, Georgia Institute of Technology, 1992.
27. BECARS software. <http://www.tsi.enst.fr/becars/index.php>.
28. Hervé Bredin, Guido Aversano, Chafic Mokbel, and Gérard Chollet. The Biosecure Talking-Face Reference System. In *2nd Workshop on Multimodal User Authentication (MMUA'06)*, Toulouse, France, May 2006.
29. D. Petrovska-Delacrétaz, G. Chollet, and B. Dorizzi, editors. *Guide to Biometric Reference Systems and Performance Evaluation*. Springer-Verlag, London, to be published.

30. BioSecure Reference Systems. <http://share.int-evry.fr/svnview-eph/>.

31. BioSecure Mutimodal Evaluation Campaign 2007 (BMEC'2007). <http://biometrics.it-sudparis.eu/BMEC2007/>.

eResearch Workflows for Studying Free and Open Source Software Development

James Howison, Andrea Wiggins, and Kevin Crowston

School of Information Studies
Syracuse University
{jhowison|awiggins|crowston}@syr.edu

Abstract. This paper introduces eResearch workflow tools as a model for the research community studying free and open source software and its development. The paper first introduces eResearch as increasingly practiced in fields such as astrophysics and biology, then contrasts the practice of research on free and open source software. After outlining suitable research data sets the paper introduces a class of tools known as scientific workflow frameworks, focusing on one—Taverna—and introducing its features. To further explain the tool a complete workflow used for original research on FLOSS is described. Finally the paper considers the trade-offs inherent in these tools.¹

eResearch refers to a set of scientific practices and technologies, sometimes called eScience or Cyberinfrastructure, which allow distributed groups of scientists to bring to bear large shared data sets, computational resources and shared workflows for scientific inquiry. A prototypical example of such research is the Upper Atmospheric Research Collaboratory (UARC) and the NSF has produced a series of reports on the topic (eg [4]). The hallmarks of eResearch are: a) broad community-level collaborations between distributed scientists, b) large-scale broadly available data sets c) shared computational analysis tools and workflows, and d) replicable research with clear provenance metadata.

While the FLOSS research community has taken some steps towards this goal we have not yet fully embraced this model of inquiry. Given that we study highly effective distributed collaborators and collaborative technologies in the FLOSS community, we have good understanding of solutions to the challenges faced by such distributed collaborations. Figure 1 shows an envisioned workflow repository which allows the discovery, replication, extension and publication of research workflows, drawing on shared components. It is time to build further towards eResearch.

1 Current FLOSS research practices

A series of papers has examined the current research practices in the investigation of FLOSS and its development [1, 2]². In general this research has been, and continues

¹ This work is supported by NSF grant #0708437. A longer version of this paper is available at <http://floss.syr.edu/publications/HowisonTavernaDemoIFIP.pdf>

² See also the report of the FOSSRRI workshop at <http://fossrri.rotterdam.ics.uci.edu/> and the Research Room @ FOSSDEM: http://libresoft.es/Activities/Research_activities/fosdem2008

Please use the following format when citing this chapter:

Howison, J., Wiggins, A. and Crowston, K., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 405–411.

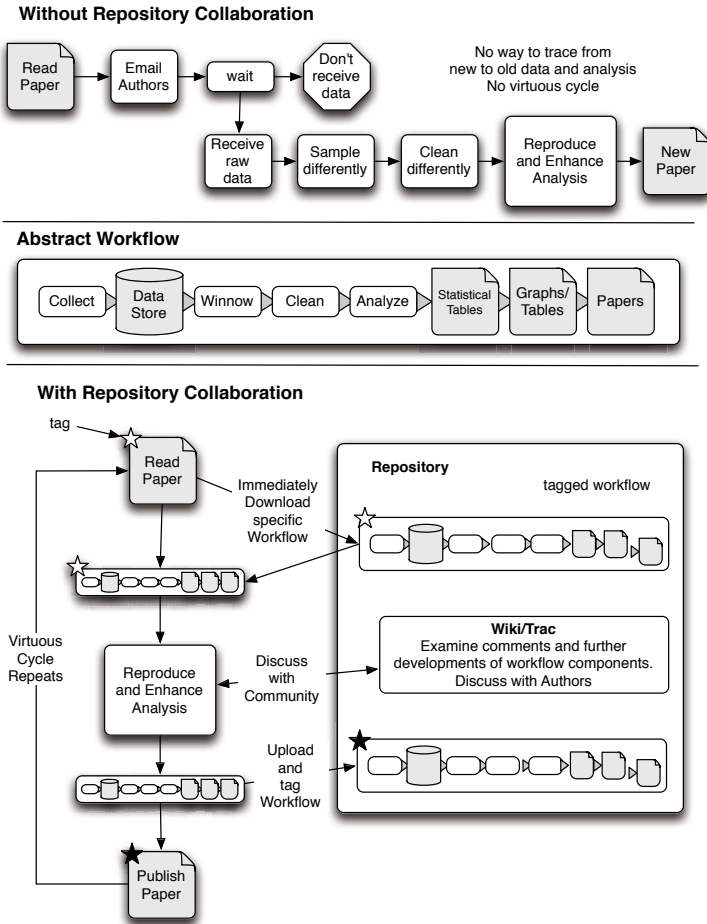
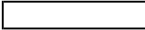


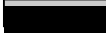


Fig. 1. Envisioned improvements in FLOSS research practices

to be, undertaken by separate groups and has involved substantial re-collection of very similar data, usually through spidering Sourceforge or similar sites.

In the past three years significant progress has been made towards shared datasets held in what have been called Repositories of Repositories [1]. At the IFIP 2.13 conference in 2007 a workshop was held for research based on public databases, including FLOSSMole and CVSanaly. In addition, the Notre Dame Sourceforge database dumps are available under an academic sub-license. These data sources provide an excellent foundation for moving research in this field towards eResearch. Figure 2 summarizes the impressive and substantial amount of data available in these RoRs, and points out gaps in data availability. Of course not all researchers use these databases, with many continuing to spider their own data sets, especially outside of such communities as IFIP 2.13 and the Mining Software Repositories workshops.

Repository for Research Data ¹ :		FLOSS mole	Notre Dame dumps	FLOSS metrics & CVSanalY	Qualoss & SQO-OSS	Source kubitizer
Project	Basic data					
	Demographics ²					
Developer Demographics	Confirmed Locations					
	Memberships					
Communication Venues	Roles					
	Mailing lists					
	Forums					
	Issue Trackers					
	IRC logs					
Software Venues	Release System					
	SVN/CVS (counts)					
	SVN/CVS full					
	Packages produced					
	Releases + Dates					
	Size (LOC, SLOC)					
	Dependencies					
Use and Popularity	Complexity Metrics					
	Downloads					
	Pageviews					
	User ratings					
	In Debian					
Sample Collected	Actual Use ³					
	Sourceforge					
	Rubyforge					
	ObjectWeb					
	Savannah (GNU)					
	Debian Distribution					
	Apache Foundation					
	GNOME meta-project					
KDE meta-project						

	Not Collected		Planned (or pilot data only)
	Partial (selected sub-collection)		Present

1. This table excludes services with data not easily available to researchers. Ohloh, for example, was excluded for this reason. The Notre Dame dumps require signing a research usage agreement. Sourcekubitizer was included insofar as it provides public access to data via the FLOSSmole project. FLOSSmetrics includes the earlier sets released by the Libre Software engineering group (CVSanalY and Debian Counts). Qualoss and SQO-OSS are included together for reasons of space, they are separate projects, but they are collaborating.

2. Project Demographics include Names, Descriptions, Founding date, Intended Audience, Operating System/environment, License, Programming language, Maturity/Status and Donors. Projects are often hosted on more than one service, or provide their own services (such as Trac, SVN etc) Confirmed Locations refers to a human effort to identify the locations actually used by each project.

3. Actual use as measured, for example, by the Debian Popularity contest which has a voluntary agent installed by some Debian users that reports frequency of package use.

4. Sourcekubitizer samples only Java projects and accepts user contributions (specify project, SCM location, homepage)

5. Qualoss intends to implement their measures on 50 projects, currently there are 5 available as pilot data. SQO-OSS works closely with the KDE meta-project.

6. FLOSSmetrics aims to have validated data for 3,000 and currently has partial data available (primarily CVSanalY) for 100 projects.

URLS: FLOSSmole: ossmole.sf.net, Notre Dame: nd.edu/~oss/, FLOSSmetrics: data.flossmetrics.com, CVSanalY: libresoft.es/Results/CVSanalY_SF, Qualoss: qualoss.org, SQO-OSS: www.sqo-oss.eu, Sourcekubitizer: sourcekubitizer.org. Thanks to Jesus González-Barahona, Gregorio Robles and Megan Conklin for assistance in preparing this table.

Fig. 2. Table showing repositories available for FLOSS research

While progress has been made in data availability there is little sharing of analyses, including components that calculate important measures such as project effectiveness. In general researchers have stuck to their (or perhaps to their graduate students) preferred data manipulation and statistical analysis tools, conducting in-house development where required. Those researchers which have made their analysis components and workflows available (such as [3] and [5]) have merely placed such bespoke tools on project websites. This paper now turns to describe tools that can move research on FLOSS towards increased collaboration and better research results.

2 Scientific workflow tools

There are a set of tools which contribute to solving the issues raised above. This section introduces two: workflow tools and a community platform for distributed collaboration around workflows.

Scientific workflow tools such as Taverna and Kepler support high-level programming which binds together data sources and analysis steps. The basic principles of the software are the same: steps in a workflow are undertaken by components which have multiple input and output ports. Components are linked by joining the output ports of one to the input ports of another. A workflow made up in such a manner can be represented simply as a flow diagram (See Figure 3, below) and is usually stored in a single XML file. As with most programming environments, much of the usefulness of these tools comes from their library of components which can be local (eg Java or *R*) or remote (eg SOAP accessed web-services). The high-level composition also promotes modularity in analysis development, believed to lead to easier collaboration and higher quality products.

Taverna The proposed demonstration focuses on Taverna and uses workflows developed to address research questions about FLOSS development. Taverna is instantiated as a stand-alone desktop application, written in Java and therefore cross-platform. Workflows can also be run via a ‘headless’ server application.

The application has two main interface modes: one for the design of a workflow and one for its execution. The design mode provides a list of available components, the workflow definition, and an automatically rendered diagram of the workflow. Input and output ports are typed through the familiar MIME typing system. For simplicity and sharing, components can be grouped into sub-workflows, with a single set of input and output ports.

Taverna supports workflows with both remote and local components. Remote services can be gathered (‘scavenged’ in Taverna parlance) by entering URLs including Web Services Description Language (WSDL), or from other workflows. It is anticipated that the development of remote services in the FLOSS domain will utilize the WSDL/SOAP combination, standard in many server technologies. Taverna parses the WSDL description file to make multiple components available, each with named input and output ports.

Local services include a library of components dealing with standard operations such as file IO, string and list manipulation. Customized local components can be written in Java, via a scripting syntax called Beanshell, and in the statistical package *R*. Data fed into a component’s input ports are available as local variables of the same

name and, similarly, output is automatically taken from variables with the same name as the output port at the end of a script. Workflows are able to incorporate typical flow of control methods, such as iteration and conditional branching. There are no global variables and components do not communicate except via their ports.

A designed workflow is executed by first providing any initial workflow inputs (such as a set of FLOSS project names or sampling criteria). An animated step-by-step process monitor summarizes analysis progress until the final outputs of the workflow are displayed. One excellent feature of the software is that during and after the execution, the full set of intermediate input and output variables can be viewed for each component, significantly improving workflow debugging and verification. An XML status report is available for each component, and the full set of status, intermediate and final results can be saved as an XML file for archiving or sharing.

Taverna provides significant metadata facilities. Firstly, a workflow or component designer can provide metadata about the workflow or individual ports, both in unstructured text descriptions and using scientific ontologies based on RDF/OWL. Secondly, the system provides a unique identifier for the workflow and a unique identifier for each and every workflow execution. These identifiers are standardized by the Object Management Group and can be used in papers to point to a specific workflow as well as the specific execution used to produce the results in the paper.

The group that developed Taverna has also developed a social networking site called MyExperiment to encourage sharing of workflows. The site allows the creation of profiles for individual researchers, groups (such as our FLOSS group), and the upload of Taverna workflows. Users can tag their contributed workflows with metadata for discovery and can download, rate, and comment on workflows. If the workflow is later used as a sub-workflow, a citation is displayed on the site.

Example workflow The authors are working to replicate a small number of FLOSS studies with Taverna workflows. These studies draw on large federated data sets (FLOSSmole, CVSanalY and the Notre Dame dumps) and will assist in the prototyping of SOAP access and a library of reusable local components. The discussion below presents a workflow prepared for a companion scientific paper [6], also published in these proceedings.

The workflow draws on FLOSSmole data to produce a time-series graph of social network centralization through a project's lifetime, based on evidence from project communications. Figure 3 shows the workflow graphic saved directly from Taverna. Workflow inputs are boxed at the top, and the final graphical output boxed at the bottom. There are three types of components used: 1) WebServices accessed via SOAP (eg GetPeriods) 2) Rshell components (eg CalculateWeight) and 3) Local components used for splitting XML results and managing iteration.

The basic flow is as follows. The user specifies a project and a start and end date, for which GeneratePeriods provides a set of overlapping periods. Then for each period, the events (dated from-to relations in project communications) are accessed from FLOSSmole by EventsForProjectInPeriod, which returns an XML document of the events. The XML is split into individual events, used by CalculateWeight to calculate a recency based edge weight, such that less recent events are lower weighted.

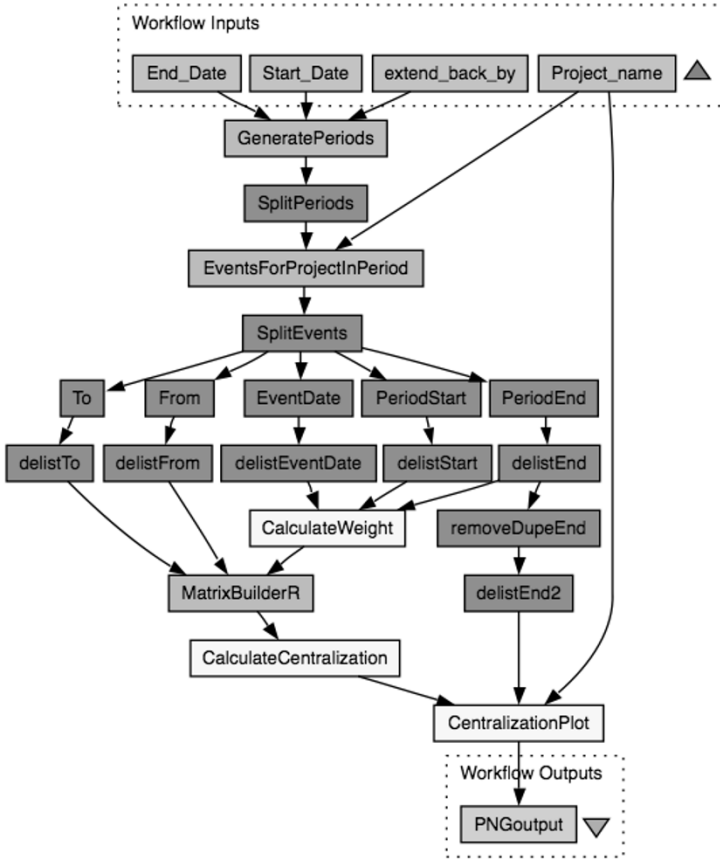


Fig. 3. An example Taverna workflow for analysis of FLOSS communications social networks over time, saved directly from the Taverna interface

The MatrixBuilderR component uses the dates and weights to generate an aggregated socio-matrix. A local *Rserve* instance uses the matrix to calculate network centralization, which is passed with an associated date to the CentralizationPlot component to produce a time-series graph and summary measures. The workflow and the workflow execution XML files that produced this diagram are available in the FLOSSmole SVN (see [6]).

3 Conclusion

The combination of growing large-scale public data sets and workflow tools such as Taverna and MyExperiment.org present a great opportunity for eResearch on FLOSS and its development. There are, of course, issues to be worked through, including a) building common interfaces to public datasets, b) creating ontologies for naming parts

of datasets, such as project and developer identifiers, c) incorporating metadata gathered about projects, such as their patch submission procedures and the location of their repositories at different times and d) incorporating social science data, such as content analytic schemas. While data sets and demonstrations are vital to encouraging researcher involvement, editors and reviewers should feel empowered to request complete workflows and insist that papers draw on available datasets, where available.

There are clearly trade-offs in standardizing on analysis technologies. For example, there is a substantial store of experience and skills with individual researcher's tools of choice. Standardization promotes collaboration but also asks research groups to move towards the standard tools, in order to benefit from the work of the collaborators. While standardization has some costs, the benefits of the collaboration it supports aren't limited to working with other research groups. Many groups have also had the experience of losing their main programmer, subsequently facing a lack of knowledge about their own systems. Indeed while it is commonly acknowledged that any form of collaboration can benefit from standardization, it is also true that programmers returning to their own work months later can also benefit from standardized approaches, enabling one to quickly build on one's own earlier work.

eResearch presents a significant opportunity for research on FLOSS development. This paper outlines what is meant by a call for a move towards eResearch techniques and describes tools and ongoing work to kickstart that process. Finally it proposes a demonstration session for the IFIP 2.13 conference in 2008 to make these ideas and tools concrete to the FLOSS research community.

References

- [1] Antoniadis I, Samoladas I, Sowe SK, Robles G, Koch S, Fraczek K, Hadzisalihovic A (2007) D1.1 study of available tools. EU Framework deliverable, FLOSS-metrics, URL http://flossmetrics.org/sections/deliverables/docs/deliverables/WP1/D1.1-Study_of_Available_Tools.pdf
- [2] Howison J, Conklin M, Crowston K (2006) FLOSSmole: A collaborative repository for FLOSS research data and analysis. *International Journal of Information Technology and Web Engineering* 1(3):17–26
- [3] Howison J, Inoue K, Crowston K (2006) Social dynamics of free and open source team communications. In: Damiani E, Fitzgerald B, Scacchi W, Scotto M (eds) *Proceedings of the IFIP 2nd International Conference on Open Source Software (Lake Como, Italy)*, IFIP International Federation for Information Processing, vol 203/2006, Springer, Boston, USA, pp 319–330, URL http://floss.syr.edu/publications/howison_dynamic_sna_intoss_ifip_short.pdf
- [4] NSF Cyberinfrastructure Council (2007) *Cyberinfrastructure vision for 21st century discovery*. URL http://netstats.ucar.edu/cyrdas/report/cyrdas_report_final.pdf, NSF Report 0728
- [5] Robles G, Amor JJ, González-Barahona JM, Herraiz I (2005) Evolution and growth in large libre software projects. In: *The 8th International Workshop on Principles of Software Evolution*, Lisbon, Portugal
- [6] Wiggins A, Howison J, Crowston K (2008) Social dynamics of FLOSS team communication across channels. In: *Proceedings of the Fourth International Conference on Open Source Software (IFIP 2.13)*, Milan, Italy

Panel: Opportunities and Risks for Open Source Software in Industry

Joseph Feller¹, Björn Lundell², Pentti Marttiin³,
Walt Scacchi⁴, and Nico Schellingerhout⁵

¹ University College Cork, Ireland, JFeller@afis.ucc.ie,

² University of Skövde, Sweden, bjorn.lundell@his.se,

³ Nokia Siemens Networks, Finland, pentti.marttiin@nsn.com,

⁴ University of California Irvine, USA, wscacchi@uci.edu,

⁵ Philips Medical Systems, The Netherlands,
nico.schellingerhout@philips.com

1 Introduction

Open Source Software (OSS) is a multi-faceted phenomenon which has become an issue of strategic importance for many commercial organisations. Stemming from an ideological issue, with emphasis on freedom and community values, we have recently seen a broader interest in the Open Source phenomenon amongst practitioners in many companies. A number of SMEs and large companies are currently exploring the potential of Open Source, and for some it has become core to their business and development activities.

Open Source is currently evolving and we have yet to see its full potential. In this panel, the goal is to stimulate discussion on the opportunities and risks to be considered with the wider adoption of Open Source systems and methods in the development of software systems in commercial organisations. In particular, the discussion will address the following issues:

- There are many different ways in which Open Source ideas can be adopted by companies, and influence the way in which companies do business and develop software systems. How can firms utilize OSS practices and tools?
- Open Source licensing has created a large commons of public software goods. How can firms create sustainable business models predicated on productizing these “commodities”?
- Open Source utilizes a legal, social and technological architecture to enable a community-based peer-production process. How can firms create sustainable business models predicated on effectively leveraging this process (either through participation or emulation)?

The panel will include short position statements by the panellists and a dialogue between them, followed by a discussion with the audience. Significant time will be allocated to an open discussion on the issues with the audience, and those attending are invited to raise their own concerns and share their own experiences.

Please use the following format when citing this chapter:

Feller, J., Lundell, B., Marttiin, P., Scacchi, W. and Schellingerhout, N., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 413–414.

2 About the Panellists

On this international panel, the five panellists are: Joseph Feller (University College Cork), Björn Lundell (University of Skövde), Pentti Marttiin (Nokia Siemens Networks), Walt Scacchi (University of California Irvine), and Nico Schellingerhout (Philips Medical Systems). The panel will be chaired by Walt Scacchi.

Joseph Feller is a Senior Lecturer in Business Information Systems, University College Cork, Ireland. He has co-authored/edited three books on open source. He chaired the IEE/ACM workshop series on Open Source Software Engineering (2001-2005), served as Program Co-Chair for the Third International Conference on Open Source Systems, and is the co-chair for the Open Source and Innovation track at ECIS 2008. Dr. Feller was a principle investigator in the EU FP6 CALIBRE project, and is currently a principle investigator in Open Code, Content and Commerce (O3C) Business Models, a three year project funded by the Irish Research Council for the Humanities and Social Sciences.

Björn Lundell has been involved in a number of collaborations on open source, including the EU FP6 CALIBRE project (2004-2006) and the industrial (ITEA) research project COSI (2005-2008). He is a founding member of the IFIP Working Group 2.13 on Open Source Software, and the founding chair of Open Source Sweden, an industry association established by Swedish Open Source companies.

Pentti Marttiin is a Manager in Nokia Siemens Networks. He has been responsible for deploying OSS based Inner Source and SCM services in Nokia and Nokia Siemens Networks. He received his PhD in information systems (1998), and holds Docent position at Helsinki School of Economics, Finland. He has actively participated in EU projects (IST-MOTION, IST-Tellmaris, ITEA-MOOSE, ITEA-COSI).

Walt Scacchi is senior research scientist and research faculty member in the Institute for Software Research, and also research director of the Computer Game Culture and Technology Laboratory, both at UC Irvine. He received a Ph.D. in Information and Computer Science at University of California, Irvine in 1981. His research interests include open source software development, computer game culture and technology, knowledge-based systems for modeling and simulating complex engineering and business processes, and software acquisition and electronic commerce/business. Last, he was Program Co-Chair of the 2006 Intern. Conf. on Open Source Systems, Como, Italy June 2006, and is General Chair of the 2007 Intern. Conf. on Open Source Systems, Limerick, IR, June 2007.

Nico Schellingerhout is a software team leader and responsible for the InnerSource initiative in Philips Healthcare. He is a task leader in ITEA COSI and has been involved in System Simulation and Software architectures research for over 10 years. He received his training in Computational Physics at the University of Groningen (M.Sc. 1988, Ph.D. 1995). His recent research interest is collaboration of distributed teams and the application of OSS practices in a corporate setting.

Open Source Environments for Collaborative Experiments in e-Science

Andrea Bosin¹, Nicoletta Dessi¹, Maria Grazia Fugini², Diego Liberati³,
and Barbara Pes¹

¹ Università degli Studi di Cagliari, Dipartimento di Matematica e Informatica,

Via Ospedale 72, 09124 Cagliari
andrea.bosin@dsf.unica.it,
{dessi,pes}@unica.it

² Politecnico di Milano, Dipartimento di Elettronica e Informazione,
Piazza da Vinci 32, I-20133 MILANO
fugini@elet.polimi.it

³ IEIT CNR c/o Politecnico di Milano, Piazza da Vinci 32, I-20133
MILANO
liberati@elet.polimi.it

Open Source Software (OSS) for e-Science should make reference to the paradigm of a distributed surrounding over a multi system mix of Web Services and Grid technologies, allowing data exchanging through services, according to standards in the area of the Grid and of Service Oriented Computing (SOC). In fact, biologists, medical doctors, and scientists are often involved in time consuming experiments and are aware of the degree of difficulty in validating or rejecting a given hypothesis by lab experiments. The benefits of OSS for e-Science consider that as many operating nodes as possible can work cooperatively sharing data, resources, and software, thus avoiding the bottleneck of licenses for distributed use of tools needed to perform *cooperative scientific experiments*. In particular, this chapter presents an architecture based on nodes equipped with a Grid and with Web Services in order to access OSS, showing how scientific experiments can be enacted through the use of a cooperation among OSS sites. Such a choice, besides reducing the cost of the experiments, would support distributed introduction of OSS among other actors of the dynamical networks, thus supporting the awareness about OSS and their diffusion. An OSS environment for cooperative scientific experiments (*e-experiments*) can effectively support the distributed execution of different classes of experiments, from visualization to model identification through clustering and rules generation, in various application fields, such as bioinformatics, neuro-informatics, tele-monitoring, or drug discovery. By applying Web Services and Grid computing, an experiment or a simulation can be executed in a cooperative way on various computation nodes of a network equipped with OSS, allowing data exchange among researchers. Our environment formalizes experiments as *cooperative services* on various computational nodes of a grid network. Basic elements are models, languages, and support tools creating a virtual network of organizational responsibility of the global experiments, according to rules under which each node can execute local services to be accessed by other nodes in order to achieve the whole experiment's results.

The OSS environment supports the simulation/execution of different classes of experiments in bioinformatics (drug discovering, micro-array data analysis,

Please use the following format when citing this chapter:

Bosin, A., Dessi, N., Fugini, M.G., Liberati, D. and Pes, B., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 415–416.

molecular docking). Each experiment ranges from visualization (browsing and search interfaces), to model identification through clustering and rules generation, and requires tools for correct design and set up of the experiment workflow and for information retrieval (e.g., for searching similar protocols, or descriptive datasheets for chemical reactors). Cooperating scientists who perform joint experiments may require specialized tools (e.g., data mining, or database tools) or computational power (e.g., long calculi for protein analysis based on their forms, or for discarding the irrelevant experiments in drug study) available only at specific nodes. The visualization part is given special attention, considering friendly interfaces and graphical simulations enabling an improved comprehension of currently textual explanations. Also privacy and data security are a major concern in our environment, considering both methods to select trusted nodes within the cooperation network, and/or to obscure or encrypt the transmitted and stored data, to preserve their sensitivity, according to user-formulated security requirements.

The environment supports experiments defined as numerical evaluations carried out on selected data sets according to available methodological approaches. The experiment execution platform is composed of: 1) a distributed workflow; 2) the involved nodes and their relative roles in the experiment; 3) the set of involved resources (data areas, data repositories and e-services). Experiments have some portions, both of processes and of data or knowledge, that can be shared in a collaborative environment. One of the reasons for executing an experiment in a distributed way might be that one organization would need to process data under a specific costly product available on a node because of its lack of skill for developing or using open source equivalent; rather than acquiring the product (e.g. SAS, ORACLE, or MATLAB), the organization might invoke a remote service as OSS available on the remote node. Another reason is that some cooperating organizations might want to inspect data dispersed on their databases, with no changes to their local computational environment. Although heterogeneous services can provide similar capabilities, the researcher is in charge of choosing the most suitable methods to accomplish each task, that is, he is in charge of designing the workflow of the scientific experiment. The run time OSS platform performs service discovery, meeting functional and non functional (e.g., price, security, quality) parameters. If the researcher wants to rerun an experiment, the workflow must take into account the changes in the choice of methods as well as in the availability of services and tools. The researcher interacts and chooses services, workflows, and data in an experimental scenario whose cooperative framework has been defined to extend the integration of scientific experiments to a level of scenario-based interaction. This scenario is profitable for many reasons, like exchanging scientific data and processing tools which results in a reduced number of software acquisitions, load balancing work between specialized researchers, and so on.

List of Demonstrations at The Fourth International Conference on Open Source Software

eResearch Workflows for Studying Free and Open Source Software Development

- James Howison, Syracuse University, US, jhowison@syr.edu
- Andrea Wiggins, Syracuse University, US, awiggins@syr.edu
- Kevin Crowston, Syracuse University, US, crowston@syr.edu

Facilitating Social Network Studies of FLOSS using the OSSNetwork Environment

- Marco A. Balieiro, Faculdade de Computação – Universidade Federal do Pará, BR, ma.balieiro@gmail.com
- Samuel F. de Sousa Júnior, Faculdade de Computação – Universidade Federal do Pará, BR, sfelixjr@gmail.com
- Cleidson R. B. de Souza, Faculdade de Computação – Universidade Federal do Pará, BR, cdesouza@ufpa.br